Broadview
www.broadview.com.cn

毫无保留

在开始学习Oracle的时候，我曾经非常困惑，“甲骨文”毕竟不好懂呀。幸运的是，我闯过了初学的难关。现在回头看真是白走了很多弯路。《Oracle数据库精讲与疑难解析》的畅销，使很多读者希望我能写一本全景介绍如何掌握Oracle应用以及数据库开发的图书。因此，我再次开始了毫无保留的写作之旅……

畅销书作者
最新力作

成功之路：ORACLE 11g 学习笔记

赵振平 编著

《Oracle数据库精讲与疑难解析》作者



视频教学

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

成功之路：Oracle 11g
学习笔记

赵振平　编著



電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书专为开发人员编写，先引导开发新手熟悉 Oracle 环境，然后进入数据库开发，并要掌握项目开发中的一些技巧。开发技巧展示是本书的一大特色（这一点很值得资深的开发人员借鉴）！

本书共 24 章，主要介绍了 Oracle 安装、干净卸载 Oracle、数据库启动关闭、Oracle 网络配置等相关内容。在介绍 Oracle 系统管理的知识以后，将介绍与开发相关的内容（如 PL/SQL 基础知识、存储过程、函数、包等），并介绍数据库性能调整，本书重点介绍 SQL 语句调优。SQL 语句调优是本书的另一大特色！

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

成功之路：Oracle 11g 学习笔记 / 赵振平编著. —北京：电子工业出版社，2010.5
ISBN 978-7-121-10781-8

Ⅰ. ①成… Ⅱ. ①赵… Ⅲ. ①关系数据库－数据库管理系统，Oracle 11g Ⅳ. ①TP311.138

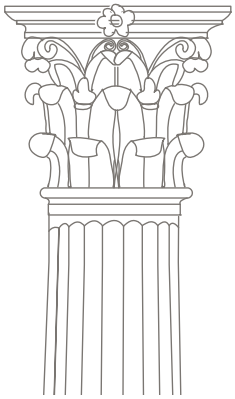
中国版本图书馆 CIP 数据核字（2010）第 076904 号

责任编辑：胡辛征
文字编辑：江 立
印 刷：北京市天竺颖华印刷厂
装 订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1092 1/16 印张：31.75 字数：700 千字
印 次：2010 年 5 月第 1 次印刷
印 数：4000 册 定价：59.00 元（含光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。



序：一份普通人的推荐

关于写序，人的一辈子有很多的第一次，给书写序，对我来说也是第一次，而且是很难得的第一次。一般人书写完了，都会找一个有点名气的人来为之做序，在这里我只能惭愧一下。因为我总觉得给一本书写序的人，在书的内容方面，至少也得是半个专家，然后才能有发言权。而本人在 Oracle 方面连半桶水都算不上，因为我只是一个开发人员而已，所以这一次也恐怕只能是“被 Oracle”一把了。

关于写书这回事，我相信在这个时代，很多人都想写，或者已经写了。但是想也好，已经写了也好，真正能把书写好的恐怕也不多。但是我们知道老赵同志的第一本书——《Oracle 数据库精讲与疑难解析》，无论是市场还是读者反馈都是非常不错的。销量也是在同类书籍中很靠前的。当然，一本书的好与不好也是一个见仁见智的问题。我们知道，在这个时代，出一本书不难，难的是要出一本好书。至于老赵同志的这本书好与不好，对开发人员会不会有所帮助，现在还不是“盖棺定论”的时候。

关于怎么学好 Oracle，说不难也难。一个新手，如果在 Oracle 领域没有个三五年时间的专心努力，恐怕是难成气候的。真可谓是不经一番冰霜苦，哪得梅花放清香。但说难也不难，古人云：“世上无难事，只怕有心人”。其实技术这个东西，不管是做 Oracle DBA 还是 Java 开发或者.NET 开发，道理都是一样的。不怕你愚笨，只要努力、专注，你就能得到你想要的。因为技术是学问，只要你不欺骗他，他是不会欺骗你的。既然你付出了，当然就会有回报。正如武侠小说中的大侠，每一个绝顶高手都是在无数个单调、枯燥、无聊的过程中成长起来的。虽然那只是艺术，但生活的道理是都一样的。老赵同志在 Oracle 领域里经历了九年多的风雨磨砺，到今天为止，也算是修成正果，在 Oracle 方面也算是个非著名的高手或者专家。如果想在 Oracle 开发或者 Oracle DBA 方面有所发展的朋友，无论是技术还是生活方面都不妨和老赵同志多做交流。

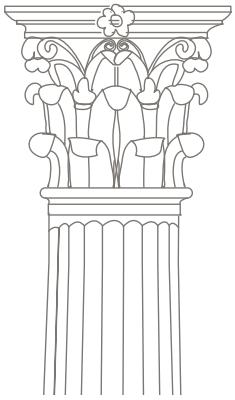
顺便再说说老赵这个人本身，虽然也有很多时候矫情得很，但总的来说还算是一个好人。做事情最大的特点就是执著和认真，曾经也有过开发经历，但是大部分时间是做 Oracle DBA，在 Oracle 领域算是个有点追求的人。本人一直以来就有个观点，老师和医生这两职业，虽然都需要学历、能力、知识，但是首先需要的还是人品。写书也一样，文章千古事，一朝永流传。人品不好的人，写出来的东西恐怕也只能遗臭千年。所以本人觉得老赵同志的书嘛，借一句流行语来说，这个可以有！

说了这么多，说说这本书吧，通过十多个日夜的努力，我读完了所有的章节。令我感受最深的是，书上提到的几乎所有问题，都是我在多年的开发过程中曾经见过或者亲身遇到过的问题。本人因为从事开发好多年了，在 Oracle 方面自然免不了遇到过各种各样的问题，为了这些问题，我曾经冥思苦想，或者在 Google 上反复搜索；当然也免不了在 QQ 上不断找老赵同志请教。现在想起来，那些问题虽然简单，但是浪费的时间可不少。类似的问题只要从事 Oracle 数据库开发的朋友，我想都会遇到。

做开发的朋友都知道，通读一本合适的书是帮助自己提高水平的最好途径之一。关于这本书我唠叨了不少，当然也只是我个人的一些想法，书本身好与不好，恐怕也不是我能说了算的。相信读者都有自己的眼光，也希望读者自己有着小马过河的精神，水深水浅自己去尝试。或许书中的知识在将来的开发过程中会对你有所帮助。

最后本人也免不了俗，我也想说些谢谢的话，但是我应该谢谁呢？我有点不知道了，谢老赵同志吧，似乎是他应该谢我，因为我帮他写序了。很多人在电视上、比赛场中或者在书里，都感谢什么兄弟、朋友甚至老婆、孩子等，但是就这本书而言，我认为最应该感谢的还是读者！

一个普通的 Java 开发者：孔向力



致读者朋友们

转瞬之间，就到了 2010 年，但人们还笼罩在金融危机的阴影之中，所谓的“专家”说：“金融危机是机遇!”机遇又何在呢？我们只能在危机中艰难前行。“艰难前行”也可以用来描述我写书的过程，经过漫长的修改和再修改，本书终于和读者朋友们见面了！

写书其实很艰难，不仅需要你的脑力劳动，还需要你的体力劳动。在喧闹的大都市，你还要能忍受寂寞和煎熬！一次一次重复的修改，好像是在考验你的意志！无论是出版社还是作者，都想给大家奉献最优秀的作品！

对于本书，我不想说“这是一本不错的书”，我只想说“在写本书时，我费了很大的力气”!

原来我也没有写本书的打算。做数据库好多年，我也接触了很多的开发人员，他们在使用数据库的过程中会遇到这样或者那样的问题。我们会经常坐下来，促膝而谈。有的毕业生会问我，如何进行数据库开发；有的开发高手会和我探讨开发技巧；更多的高手会传授我一些技巧。如果是新手请教我，我会尽量满足他们的要求（其实满足别人的要求也是一种学习）；如果遇到高手，我很乐意和他们一起探讨，因为从他们那儿，我能学到很多很多。其实，无论是新手还是高手，他们都可以算得上是我的老师！

正是和他们的接触，我才明白新手会遇到什么问题、新手最关注什么问题，以及高手的焦点是什么，这些问题正是写作本书的源泉。与其说本书是我写的，还不如说是他们教我写的。

从他们那儿我明白，新手最关注数据库的使用（而无须过分关注数据库的原理）及使用中出现问题的解决方法，而高手最关注开发技巧，这正是本书的主旋律！

本书特点

1．专为开发人员编写

本书站在开发人员的角度，为开发人员着想。如果你是一位数据库开发人员或者你立志成为软件开发人员，本书值得参考。

2．配套光盘——让你学习更轻松

如果你觉得阅读书本很无聊，请泡杯茶，找个沙发，坐下来，慢慢学习本书光盘中的

视频，它让你的学习更加 Easy！视频不仅仅是书中内容的重复，更是书中内容的有益补充。如果你想轻轻松松学习 Oracle，可千万不要错过光盘内容哦。

3．循序渐进——遵循开发人员的成长规律

本书有点类似于开发人员的成长史。对于一个新手来说，先要熟悉 Oracle 的环境（也就是基本的系统管理），新手遇到的问题，本书都会重点进行讲解；熟悉 Oracle 的环境后，我们将把你带到数据库开发的天地，并给你演示项目开发中的技巧。

4．开发技巧展示——让你在实际工作中得心应手

也许你学习了很多知识点，但是仍然开发不出你想要的东西，可能是你还没有掌握开发过程中的一些技巧，本书将会讲解实战中的一些技巧（请参考第 16 章等章节）！

5．SQL 语句调优——让你技高一筹

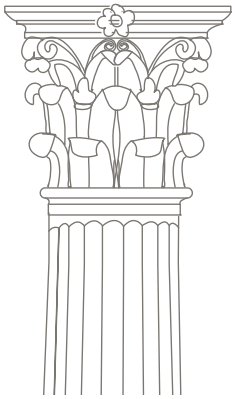
虽然你是有多年开发经验的开发人员，但是你不一定能够写出高速运行的 SQL 语句。本书将告诉你什么是高速运行的 SQL 语句，以及如何写出高速运行的 SQL 语句（请参考第 17 章）。

致谢

在编写本书的过程中，我得到了许多人的支持，包括我的家人（母亲、姐姐、姐夫、弟弟、姑姑等）和朋友（李才应、李小明、张文娟、唐志军、王冬梅、高阳、周新忠、吴小牛、王贺尊、赵翠竹、徐平平、陈猛、刘贵丽）。对于本书，易应仔先生也做出了巨大的贡献，在此表示感谢！另外还要感谢的是电子工业出版社的胡辛征老师、江立小姐、李玲小姐以及其他工作人员。衷心感谢大家！

作者：赵振平

2010 年 4 月 20 日



目录

第 1 章　你真正学会数据库了吗 ……1

万事开头难，现在，我们将进入 Oracle 的世界。什么是数据库、为什么要学习数据库、怎样才算真正学会了数据库、什么是 E-R 图、什么是数据字典……读完本章，你也许会找到想要的答案。

1.1　学习数据库的必要性 ……1
1.2　学习数据库的境界 ……2
1.3　Oracle 霸主地位之——为什么要选择学习 Oracle ……2
1.4　数据库（Database） ……2
1.4.1　何为数据库 ……2
1.4.2　何为关系数据库（RDB） ……2
1.4.3　何为数据库管理系统（DBMS） ……3
1.4.4　何为 E-R 图（Entity-Relationship Model） ……3
1.4.5　何谓表（Table） ……4
1.5　数据库在软件架构中的地位 ……5
1.6　数据库中的数据库——数据字典 ……5
1.6.1　数据字典的结构 ……5
1.6.2　数据字典的使用 ……6
本章总结 ……6

第 2 章　数据库安装不求人之——自个安装 Oracle 数据库 ……7

Oracle 的安装是一个老生常谈的问题，但又是一个不得不讨论的话题。本章介绍 Oracle 服务器端和客户端的安装，并配有安装视频。安装视频是本书的有益补充！伴随着 Oracle 新版本的不断发行，Oracle 的安装会有略微的差异，请查看 Oracle 的官方文档。

2.1　Oracle 服务器软件安装之——Linux 平台 ……7
2.1.1　准备操作系统 ……7
2.1.2　启动 Oracle 安装 ……15

2.2 Oracle 服务器软件安装之——Windows 平台……16
2.2.1 Oracle 服务器软件安装……16
2.2.2 Oracle 服务器软件安装之——故障排查……21
2.2.3 创建 Oracle 11g 数据库（Windows 平台）……22
2.2.4 Oracle 安装后期——测试安装是否成功……32
2.3 Oracle 客户端安装……33
本章总结……34
第 3 章 初学者最麻烦的问题之——不能干净地卸载 Oracle……35
为什么我们要“干净”地卸载 Oracle 呢？正是因为重装 Oracle 时，经常会遭遇失败。所谓“干净”，就是把 Oracle 从操作系统中清除得彻彻底底！有人说，“卸载”是一件再容易不过的事情了，没有必要拿出来小题大做…错也！…错也！…按照 Oracle 的官方文档，你就是不能“干净”地卸载 Oracle！
3.1 Oracle 不能完全卸载的烦恼……35
3.2 干干净净地卸载 Oracle 数据库……35
3.2.1 用 DBCA 删除数据库……36
3.2.2 删除 ASM……37
3.2.3 停止 Oracle 所有的服务（Oracle Services）……38
3.2.4 用 Oracle Universal Installer 自动删除 Oracle 的组件……38
3.2.5 手动删除 Oracle 的其余组件……38
本章总结……39
第 4 章 轻松搞定 Oracle 网络……40
本章将介绍与 Oracle 相关的网络问题。Oracle 网络建立在操作系统之上。配置操作系统网络是配置 Oracle 网络的第一步。在配置 Oracle 网络之前，我们必须确保操作系统的网络配置无误。当客户端不能连接数据库服务器时，笔者将教你一步一步地排除网络故障。
4.1 轻松搞定 Oracle 网络之——数据库服务器端网络配置……40
4.1.1 监听器的配置……41
4.1.2 监听器的启动……44
4.1.3 监听器的关闭……46
4.1.4 监听器的状态……47
4.2 轻松搞定 Oracle 网络之——Oracle 客户端网络配置……48
4.2.1 客户端配置……48
4.2.2 测试客户端到服务器的连接……56
4.2.3 登录远程数据库……56
4.3 轻松搞定 Oracle 网络之——Oracle 网络故障攻略……57
本章总结……59
第 5 章 访问远程数据库之——工具介绍……60

要想访问数据库，就需要用到访问数据库的各种客户端工具。本章主要介绍如何使用各种客户端工具访问远程数据库。这些客户端工具包括 SQL*Plus、企业管理器 OEM 和一些第三方的工具(如 DbVisualizer、TOAD)。

5.1 字符界面管理工具之——SQL*Plus ····60
5.1.1 启动 SQL*Plus ····60
5.1.2 在 SQL*Plus 中执行代码 ····61
5.1.3 关闭 SQL*Plus ····63
5.1.4 最有用的 SQL*Plus 命令（SQL*Plus Commands） ····63
5.1.5 SQL*Plus 使用技巧 ····66
5.2 Web 版的企业管理器（Oracle Enterprise Manager，OEM） ····73
5.2.1 使用 OEM ····74
5.2.2 启动 dbconsole ····75
5.2.3 关闭 dbconsole ····76
5.3 其他数据库访问工具 ····76
5.3.1 DbVisualizer ····76
5.3.2 TOAD ····77
5.3.3 Embarcadero SQL Tuner ····77
5.3.4 PL/SQL Developer ····77
本章总结 ····79
第 6 章 Oracle 数据库启动/关闭——原理剖析 ····80

数据库启动、关闭的过程其实很简单，但是原理却比较复杂。本章的重点是理解 Oracle 启动和关闭的原理。对于开发人员来说，也许没有必要深度理解数据库启动、关闭的原理。但是，我们需要知道如何启动、关闭数据库。

6.1 数据库的启动 ····80
6.1.1 启动一个关闭的数据库（常规启动） ····80
6.1.2 数据库启动原理 ····82
6.1.3 初始化参数（Initialization Parameter） ····83
6.1.4 只启动实例——把数据库启动到非装载状态（NOMOUNT） ····89
6.1.5 把数据库启动到装载状态（MOUNT） ····90
6.1.6 打开数据库 ····90
6.1.7 重启你的数据库（Reset） ····92
6.1.8 把数据库启动到只读模式 ····92
6.1.9 把数据库设置成自动启动/关闭之——Windows 篇 ····92
6.1.10 把数据库设置成自动启动/关闭之——UNIX/Linux 篇 ····94
6.2 数据库启动故障排查攻略 ····97
6.3 数据库的关闭 ····100

6.3.1 使用频率最高的数据库关闭方式 ····· 100
6.3.2 数据库关闭最快的方式 ····· 101
6.3.3 数据库关闭最慢的方式 ····· 101
6.3.4 以 TRANSACTIONAL 方式关闭数据库 ····· 102
本章总结 ····· 102

第 7 章 数据之家——Oracle 存储管理 ····· 103

表空间、数据文件、块、区、段等这些概念，也是笔者以前曾经模糊的问题。本章重点介绍表空间和数据文件，并理解 Oracle 存储的物理结构和逻辑结构。看图学习 Oracle 的存储结构，是理解 Oracle 存储结构的捷径。

7.1 常规表空间管理 ····· 103
7.1.1 创建表空间（Create Tablespace ） ····· 103
7.1.2 扩展表空间 ····· 108
7.1.3 使表空间离线/在线（Taking Tablespaces Offline/Online） ····· 108
7.1.4 重命名表空间（Renaming Tablespaces） ····· 109
7.1.5 使表空间只读（Read Only） ····· 109
7.1.6 删除表空间（Drop Tablespace） ····· 110
7.1.7 大文件表空间（Bigfile Tablespaces） ····· 110
7.1.8 表空间的加密（Encrypted Tablespaces） ····· 111
7.1.9 表和表空间的关系 ····· 111
7.1.10 必须清楚的几个概念 ····· 113
7.2 数据文件管理 ····· 115
7.2.1 使数据文件离线 ····· 115
7.2.2 移动数据文件 ····· 116
7.2.3 删除数据文件 ····· 117
7.2.4 列出数据库中的数据文件 ····· 118
7.3 临时表空间管理 ····· 118
7.4 回滚表空间管理 ····· 122
7.4.1 回滚数据（Undo Data） ····· 122
7.4.2 读一致性（Read Consistency） ····· 122
7.4.3 数据库恢复（Database Recovery） ····· 123
7.4.4 闪回查询（Flashback Query） ····· 123
7.4.5 事务和回滚数据（Transactions and Undo Data） ····· 123
7.4.6 回滚段（Undo Segment） ····· 123
7.4.7 回滚表空间（Undo Tablespace） ····· 125
7.4.8 回滚表空间的操作 ····· 126
本章总结 ····· 128

第 8 章　简化存储管理之——自动存储管理（ASM）……129

自动存储管理（Automated Storage Management，ASM）是 Oracle 10g 推出的新功能，它实质上是一个卷管理器，用于替代操作系统所提供的逻辑卷管理器（LVM），对磁盘进行有效管理。本章重点介绍 ASM 的原理及配置。在 Oracle 11g 中，Oracle 公司强烈推荐使用 ASM 做存储管理。

8.1　ASM 实例（ASM Instances）……129
8.2　ASM 磁盘组（ASM Disk Groups）……129
8.3　ASM 文件（ASM Files）……130
8.4　区（Extents）……131
8.5　镜像和失败组（Mirroring and Failure Groups）……131
8.6　ASM 使用全程……132
8.6.1　配置 ASM 实例……132
8.6.2　ASM 实例的连接……140
8.6.3　ASM 的启动……140
8.6.4　ASM 的关闭……140
8.6.5　创建磁盘组……141
8.6.6　在磁盘组上创建表空间……142
8.7　ASM 磁盘组的维护……143
本章总结……144

第 9 章　为数据库优化打基础之——Oracle 内存管理……145

前文中我们介绍了实例，实例由内存和后台进程组成。虽然 Oracle 的内存管理越来越趋于自动化，但是我们仍然需要了解 Oracle 的内存结构。本章将介绍 Oracle 的内存结构，并重点介绍数据库高速缓冲区（Database Buffer Cache）、重做日志缓冲区（Redo Log Buffer）、共享池（Shared Pool）、大池（Large Pool）、Java 池（Java Pool）、流池（Streams Pool）。

9.1　Oracle 内存介绍……145
9.1.1　系统全局区（SGA）……145
9.1.2　程序全局区（PGA）……148
9.1.3　软件代码区（Software Code Areas）……150
9.2　Oracle 内存管理方法……150
9.2.1　启用自动内存管理（Automatic Memory Management）……151
9.2.2　启用自动 SGA 管理（Automatic Shared Memory Management）……153
9.2.3　启用自动 PGA 管理（Automatic PGA Memory Management）……154
9.2.4　SGA 手动管理……155
9.2.5　PGA 手动管理……156
9.3　内存信息查询……156
9.3.1　从命令行查看 Oracle 的内存信息……156

9.3.2 用 Oracle 企业管理器查看 Oracle 内存的分配情况 ····· 160
本章总结 ····· 161

第 10 章 Oracle 的发动机——进程 ····· 162

后台进程是 Oracle 实例的组成部分，我们已经介绍过 Oracle 的内存结构，本章将介绍 Oracle 的进程结构，并重点介绍 Oracle 的后台进程及进程管理。在介绍进程时，我们会把内存、进程、物理文件结合在一起进行说明。

10.1 进程简介 ····· 162
10.2 Oracle 后台进程（Background Process） ····· 165
10.3 进程管理与维护 ····· 169
10.3.1 查看 Oracle 的后台进程 ····· 169
10.3.2 查看数据库中的会话 ····· 170
10.3.3 最热门的话题——提高数据库最大会话数 ····· 170
10.3.4 终止会话（Terminating Sessions） ····· 171
10.3.5 修改归档进程（ARCn）的数量 ····· 172
10.3.6 修改数据库书写进程（DBWn）的数量 ····· 174
10.3.7 修改作业队列进程（Job Queue Process）的数量 ····· 174
10.3.8 查看每个进程占用的系统资源 ····· 174
本章总结 ····· 176

第 11 章 数据库物理结构信息的仓库——Oracle 控制文件 ····· 177

控制文件是一个 DBA 最害怕丢失的文件。本章将对控制文件进行介绍。学习控制文件时，应该理解控制文件的内容及其重要性，并区别控制文件与其他 Oracle 物理文件。要记住，对控制文件的最好保护就是镜像控制文件。

11.1 控制文件（Control Files）简介 ····· 177
11.2 查看数据库中的控制文件 ····· 178
11.3 DBA 必做的事情——镜像控制文件 （Multiplex Control Files） ····· 179
11.4 查看控制文件的内容 ····· 180
11.5 得到创建控制文件的脚本 ····· 180
11.6 移动控制文件 ····· 181
11.7 删除控制文件 ····· 182
本章总结 ····· 182

第 12 章 服务器掉电亦无忧之——Oracle 重做日志（Redo Log）管理 ····· 183

一个漆黑的夜，狂风、暴雨……全城停电，一台高速运转的大型数据库服务器也戛然而止。公司所有的业务被迫停顿……几名工程师行色匆匆地赶到机房……来电了……他们给机器加电……重启数据库服务器……一切正常！

对于这样的事故，DBA 恐怕已习以为常。Oracle 有一套机制保证 DBA 高枕无忧，这其中最大的功劳非“重做日志”莫属！重做日志（Redo Log）保证在服务器掉电的情况下，也不会发生数据的丢失，这正是重做日志的神奇之处！

12.1 重做日志文件的工作原理……183
12.1.1 重做日志（Redo Log）……183
12.1.2 重做日志条目（Redo Entry）……183
12.1.3 重做日志（Redo Log）的构成……183
12.1.4 重做日志（Redo Log）的工作原理……184
12.1.5 重做日志文件（Redo Log Files）的四种状态……185
12.1.6 同一日志组中成员的分布……185
12.1.7 重做日志（Redo Log）与实例恢复……185
12.2 查询数据库中的重做日志组……186
12.3 查询数据库中的重做日志文件……186
12.4 创建重做日志组（Creating Redo Log Groups）……187
12.5 添加重做日志文件（Creating Redo Log Members）……187
12.6 删除重做日志组（Dropping Redo Log Groups）……187
12.7 删除重做日志成员（Dropping Redo Log Members）……188
12.8 改变重做日志组的状态……188
12.9 查看重做日志组是否已经归档……189
本章总结……189

第 13 章 总揽全局——Oracle 体系结构综述……190

遨游在数据库的海洋中，我们好似海滩边捡贝壳的小孩子，今天看到一个小贝壳，漂洗一下，收入囊中；明天发现镶嵌在泥中的小贝壳，总是刨根问题；有时在海边漫步，不经意间踢到被别人丢弃的小贝壳，我们也禁不住驻足观看。偶尔看到小贝壳在蔚蓝的大海中随波逐流，我们也会不假思索地去追逐和探索！我们需要冒险，我们需要创新，我们需要改变！

在以前的学习中，我们收集了太多零零散散的小贝壳，散而无用！只有把这些小贝壳串联起来，深度加工，才能体现它们的价值。学习任何技能也都如此，只有融会贯通，才能技高一筹！不过，这只是笔者的愚见！所以，本章将遍历 Oracle 的知识点，以帮助大家实现融会贯通。

13.1 物理数据库结构（Physical Database Structures）……191
13.2 逻辑数据库结构（Logical Database Structures）……191
13.3 实例（Database Instance）……191
13.3.1 SGA……192
13.3.2 后台进程（Background Processes）……192
13.3.3 最容易混淆的两个概念——数据库和实例……192
13.4 阅读参考……194
本章总结……194

第 14 章　成为数据库的王者之——权限管理 ····· 195

若想在数据库中无拘无束地翱翔，必须取得数据的控制权，本章将和大家一起探讨如何成为“数据库的王者”! 本章主要介绍用户、角色、权限以及这三者之间的关系，初学者应该好好理解角色和权限的关系。

14.1　用户管理 ····· 195
14.1.1　创建用户账号 ····· 195
14.1.2　预定义账户（Predefined User Accounts） ····· 197
14.1.3　修改用户账号属性 ····· 197
14.1.4　锁定爱捣蛋的用户 ····· 198
14.1.5　查看数据库账户属性 ····· 198
14.1.6　让一个用户从数据库中永远消失 ····· 200
14.1.7　查看当前登录数据库的用户 ····· 200
14.2　限制用户对资源的消耗——概要文件 （User Profiles） ····· 201
14.2.1　创建概要文件 ····· 201
14.2.2　修改用户的概要文件（把概要文件赋予用户） ····· 202
14.2.3　删除概要文件 ····· 202
14.3　数据库管理员的验证方式（Authentication Method for Database Administrators） ····· 203
14.3.1　管理员的验证之一——数据字典验证（Data Dictionary Authentication） ····· 203
14.3.2　管理员的验证之二——使用操作系统验证（Using Operating System Authentication） 203
14.3.3　管理员的验证之三——使用口令文件验证（Using Password File Authentication） ····· 204
14.3.4　管理员的验证之四——基于网络的验证 ····· 207
14.4　权限管理 ····· 207
14.4.1　力量的象征——权限 ····· 207
14.4.2　搞不清楚的问题——角色与权限 ····· 213
14.4.3　创建角色 ····· 214
14.4.4　授予角色权限 ····· 214
14.4.5　删除角色 ····· 214
14.4.6　权力演绎 ····· 214
本章总结 ····· 217

第 15 章　Oracle 数据泵（Oracle Data Pump） ····· 218

在信息化的时代，数据从一种载体转移到另一种载体，从一种形式变换到另一种形式，需要从一个平台移动到另一个平台。数据的存储格式有文本文件、Excel、Access 等。我们常常需要把数据从文本文件导入 Oracle，或者把 Oracle 中的数据移动到 SQL Server，或者把 Excel 的数据库装载到 Oracle。各个数据库厂商几乎都提供了数据迁移的工具，第三方厂商也提供了更加便捷的数据迁移工具。本章重点介绍 Oracle 数据迁移工具——Oracle 数据泵（Oracle Data Pump）。

15.1 Oracle 数据泵的组件……218
15.2 数据泵移动数据的方式……219
15.3 数据泵的工作原理……219
15.4 目录问题……220
15.5 导入导出过程的监控……220
15.6 数据泵导出工具（Data Pump Export）……221
15.6.1 Data Pump Export 的调用接口……221
15.6.2 导出模式（Data Pump Export Modes）……221
15.6.3 导出参数说明……224
15.7 数据泵导出工具（Data Pump Import）……231
本章总结……234

第 16 章 学习任何数据库必学之——SQL 语句……235

本章将展示开发过程中 SQL 语句的使用技巧，这是本书的亮点！本章将介绍数据库中各种类型的表，并重点介绍 SQL 语句。因为追求技巧性，所以本章耗费了笔者大量的时间，不过，如果读者愿意和我一起分享，那么所有付出都是值得的！

16.1 SQL 简述……235
16.2 事务（Transaction）……237
16.2.1 提交（COMMIT）……237
16.2.2 回滚（ROLLBACK）……238
16.2.3 事务命名（SET TRANSACTION NAME）……238
16.2.4 事务读一致性（Transaction-level Read Consistency）……239
16.2.5 回滚点（SAVEPOINT）……240
16.2.6 自治事务（Autonomous Transactions）……241
16.3 数据类型（Oracle Data Type）……241
16.4 数据库的核心——表结构管理……242
16.4.1 Oracle 的表类型……242
16.4.2 建表——创建存储数据的“容器”……244
16.4.3 修改表——修改“容器”的属性……249
16.4.4 表管理小技巧……251
16.4.5 删除表——让整个表从数据库中彻底消失……256
16.5 数据库的核心——表数据管理……257
16.5.1 往“容器”中装入数据（INSERT 语句演示）……257
16.5.2 修改“容器”中的数据（UPDATE 语句演示）……263
16.5.3 查询“容器”中的数据（SELECT 语句演示）……265
16.5.4 删除“容器”中的数据（DELETE 语句）……303
16.6 数据库的核心——约束管理……306

16.6.1 五大约束……306
16.6.2 约束管理……307
16.7 数据库的核心——LOB 管理……312
16.7.1 BFILE……313
16.7.2 BLOB……313
16.7.3 CLOB……313
16.7.4 NCLOB……314
16.7.5 BFILE 应用实战……314
16.7.6 CLOB 应用实战……315
16.8 用户自定义数据类型（User-defined Data Type）……317
16.8.1 对象类型（Object Type）……317
16.8.2 可变数组类型（Variable-size Arrays Type）……319
16.8.3 嵌套表类型（Nested Table Type）……322
16.9 分布式应用……325
16.9.1 数据库链接分类……325
16.9.2 创建数据库链接……326
16.9.3 数据库链接的应用……326
16.9.4 删除数据库链接……326
16.10 分区表管理……327
本章总结……331
第 17 章 构筑高速运行的 SQL 语句……332

你肯定会写 SQL 语句，也能写出复杂的 SQL 语句，但是你不一定能够写出高效的 SQL 语句。做 DBA 的都知道，一条效率低下的 SQL 语句足以让你的整个系统崩溃（看似死机）。本章笔者将和你一起构造高速运行的 SQL 语句。

17.1 数据库性能调整简介……332
17.2 实例调整……332
17.2.1 自动工作负载库（Automatic Workload Repository，AWR）……333
17.2.2 性能监测器（Automatic Database Diagnostic Monitor，ADDM）……334
17.3 SQL 语句调整……336
17.3.1 数据查询的加速器——索引……337
17.3.2 优化器（Optimizer）……340
17.3.3 SQL 语句执行的大致过程……340
17.3.4 SQL 语句的优化过程……341
17.3.5 提示（Hint）……352
本章总结……355
第 18 章 模式对象管理……356

本章将主要介绍模式对象。学习模式对象时，不要耗费大量的时间去记忆语法，应该集中精力于模式对象的理解。模式对象包括表（Tables）、视图（Views）、聚簇（Clusters）、约束（Constraints）、数据库链接（Database Links）、触发器（Database Triggers）、维（Dimensions）等。

18.1 数据的封藏——视图……357
18.1.1 视图创建……357
18.1.2 在存储过程中创建视图……358
18.1.3 视图的使用……358
18.1.4 得到创建视图的 SQL 语句……359
18.1.5 视图的删除……359
18.2 产生唯一值——序列（Sequence）……360
18.2.1 创建序列……360
18.2.2 序列的使用……361
18.2.3 序列值丢失（不连续）的问题……362
18.2.4 把序列的值重新置成 1……362
18.2.5 删除序列……362
18.3 隐藏数据库对象的名字——同义词（SYNONYM）……363
18.3.1 同义词的优点……363
18.3.2 同义词的分类……364
18.3.3 创建同义词……364
18.3.4 使用同义词……364
18.3.5 删除同义词……364
本章总结……365

第 19 章 开发基础的沉淀——PL/SOL 程序设计……366

本章将介绍如何利用 PL/SQL 开发数据库应用程序。如果你立志做 Oracle 的开发，请你认真阅读本章。本章重在介绍开发，而弱化系统管理。本章介绍的内容包括：PL/SQL 的基本原理及基本语法、游标、触发器及动态 SQL。尤其是动态 SQL，它是开发人员关注的热点问题。

19.1 PL/SQL 介绍……366
19.1.1 PL/SQL 基本要素……367
19.1.2 PL/SQL 块（PL/SQL Block）……368
19.1.3 变量和常量……369
19.1.4 数据类型……372
19.1.5 PL/SQL 表达式和运算符……375
19.1.6 PL/SQL 语句……376
19.1.7 在 PL/SQL 中使用 SELECT 语句……376
19.1.8 在 PL/SQL 中使用 DML……376
19.1.9 控制结构……377

19.1.10 GOTO 语句……380
19.1.11 异常（Exception）……380
19.2 数据批处理——游标（Cursor）……384
19.2.1 游标的分类……384
19.2.2 游标 FOR 循环……386
19.2.3 向游标中传递参数……387
19.2.4 游标变量……388
19.3 一触即发——触发器介绍……390
19.3.1 触发器的功能……390
19.3.2 触发器的组成……391
19.3.3 触发器的种类……391
19.3.4 触发动作（Trigger Action）的编码……394
19.3.5 触发器的维护……396
19.3.6 在触发器中，如何利用学生表 1 的值去更新学生表 2 的值……399
19.3.7 阻止触发语句的执行……401
19.4 DDL 语句的救星——动态 SQL （Dynamic SQL）……402
19.4.1 静态 SQL（Static SQL）……403
19.4.2 动态 SQL（Dynamic SQL）……403
19.4.3 动态 SQL 和静态 SQL 的比较……404
19.4.4 动态 SQL 的书写……404
本章总结……414
第 20 章 存放在数据库服务器中的代码……415

存放在数据库服务器中的代码以其高速执行备受程序员的推崇！本章主要介绍存储过程、函数、包等。创建存储过程、函数、包的语法其实并不重要，重要的是理解三者的原理及用途。同时，我们需要理解三者的区别。

20.1 存储 PL/SQL 单元……415
20.2 子程序的参数……415
20.3 存储过程（Procedure）……416
20.3.1 存储过程的创建……417
20.3.2 存储过程的调用……418
20.3.3 存储过程的编译……418
20.3.4 存储过程的删除……418
20.3.5 存储过程使用技巧……419
20.4 函数（Function）……425
20.4.1 函数的创建……425
20.4.2 函数的调用……426

20.4.3 函数的编译……427
20.4.4 函数的删除……427
20.4.5 函数使用技巧……427
20.4.6 常用函数……430
20.5 包（Package）……432
20.5.1 包的优点……432
20.5.2 包的组成……433
20.5.3 包中对象的私有属性和公有属性……433
20.5.4 包的创建……433
20.5.5 引用包的对象（元素）……435
20.5.6 让其他用户使用你的包……436
20.5.7 包的编译……436
20.5.8 包的删除……436
20.5.9 包中存储过程和函数的重载……437
20.5.10 Oracle 内置包……439
20.5.11 查看包中的存储过程或者函数……440
20.6 存储过程和函数的调试（Debug）……440
本章总结……441

第 21 章 Oracle 即时客户端（Instant Client）……442

Oracle 即时客户端（Instant Client）是一个 Oracle 公司发布了好多年，但又被大部分开发人员忽略了好多年的，然而很值得推荐的好东东！它让你的应用程序部署变得非常容易，不信就试试看吧！

21.1 即时客户端简介……442
21.2 使用即时客户端部署应用程序……443
本章总结……444

第 22 章 数据库的备份……445

备份和恢复是 DBA 的重要职能，Oracle 提供的备份和恢复机制功能非常强大，它能在数据库发生灾难的情况下，也能把数据库恢复到灾难前的状态。本章主要介绍备份的基本原理，并介绍 Oracle 提供的备份工具 RMAN。

22.1 数据库故障的种类……445
22.1.1 实例失败……445
22.1.2 介质失败……446
22.1.3 用户错误……446
22.2 冷备份（Cold Backup）……446
22.2.1 一致性冷备份（Consistent Backup）……446
22.2.2 不一致冷备份（Inconsistent Backup）……446

22.3 热备份（Hot Backup）……446
22.4 备份、恢复工具……447
22.4.1 用户管理的备份和恢复……447
22.4.2 RMAN……447
22.5 冷备份的实施……450
22.6 热备份的实施……451
22.6.1 热备份的前提——归档管理……451
22.6.2 备份整个数据库……453
22.6.3 备份单个表空间……454
22.6.4 备份单个数据文件……454
22.6.5 备份归档日志文件……454
22.6.6 备份控制文件……455
22.6.7 备份 SPFILE……455
本章总结……456
第 23 章 数据库的恢复……457
恢复能够帮助你找回丢失的数据。但备份是恢复的前提，制定完备的备份策略才是 DBA 最优先考虑的问题！本章将教会你如何恢复整个数据库、单个表空间、单个数据文件以及损坏的数据块。
23.1 数据冲突的修复……457
23.2 冷备份的恢复……458
23.3 热备份的恢复……459
23.3.1 恢复整个数据库……459
23.3.2 恢复单个表空间……460
23.3.3 恢复单个数据文件……460
23.3.4 修复数据块……461
本章总结……461
第 24 章 项目实战之——网银子系统……462
不深入实际就没有发言权，在前面的章节中，我们准备了很多理论知识，只期盼有一天能够派上用场！在这里，我们将给你一个实战的机会！在学习软件开发的时候，实践是必不可少的一个环节，正所谓“实践出真知”!
24.1 软件生命周期（System Development Life Cycle）……462
24.2 业务需求分析——前期工作……463
24.2.1 业务需求……463
24.2.2 系统效果展示……463
24.3 架构设计——架起业务与软件的桥梁……464

24.3.1 软件系统框架设计 ····· 464
24.3.2 数据库模型设计 ····· 465
24.4 编码 ····· 467
24.5 测试 ····· 467
24.6 把理想变成现实 ····· 467
24.6.1 安装 JDK ····· 467
24.6.2 安装 Tomcat ····· 470
24.6.3 网银子系统配置 ····· 474
24.7 代码解释 ····· 476
24.7.1 安装 Eclipse ····· 476
24.7.2 Eclipse 导入项目 ····· 476
24.7.3 MVC 框架 ····· 477
24.7.4 重要代码讲解 ····· 478
本章总结 ····· 484

第 1 章　你真正学会数据库了吗

小 A，大学刚毕业，会用图形界面工具创建表、删除表以及往表中插入数据。这是真正学会数据库了吗？听起来好像在操作 Excel 表格！

老 K，毕业多年，混迹于中关村的高科技公司，熟悉数据库，做过大型 MIS 项目，能写复杂的 SQL 语句，可是，老 K 开发的系统总是比别人慢半拍。老 K 真的学会数据库了吗？

老 M，懂得 Java 开发，做过 VC 开发，可是老 M 写的 SQL 语句一放到生产环境中，就会把数据库服务器搞死，老板郁闷不已，老 M 也郁闷不已！但是，其他同事却在背后羡慕老 M 的“伟大”！老 M 真正学会数据库了吗？



1.1　学习数据库的必要性

2008 年 1 月 15 日，花旗银行公布财报，其 2007 年第四季度单季度亏损 98 亿美元，这是花旗银行 196 年历史上最大的单季度亏损。由此，由美国次贷危机引发的世界金融危机全面爆发，向世界其他主要发达国家和一些发展中国家不断蔓延扩散，全球主要金融市场急剧动荡，股指连续多日大幅下挫。蓬勃发展的中国经济也受到重创。中国的各大公司纷纷裁员，校园招聘寒意阵阵。曾经的天之骄子们也被席卷进这股寒流。毕业了，还是失业了？银行业的萧条也波及了 IT 业，各大 IT 公司也在风雨中飘摇。但是，无论风有多凶，雨有多大，风雨总会过去。在风雨过去之前，我们必须积蓄力量，等待风雨过后的爆发。对于即将步入 IT 业的大学生来说，也许现在正是“老骥伏枥”的时候。

如果你想搞软件开发，如果你想成为 DBA（数据库管理员），请耐心听我絮叨几句。对于 DBA 来说，无可厚非，你必须学习数据库——那是你的天职。对于开发人员来说，也许你觉得学习数据库并不那么重要，但是，坦白地说，数据库开发通常是软件开发的一部分，也是你的工作重心之一，你说，我们应该学习数据库吗？更为重要的是，如果你想写出高效的 SQL 语句，更有必要深层次地理解数据库。一条效率低下的 SQL 语句足以让你的系统瘫痪（这句话一点都不夸张！）。

自从有了人类那一天起，我们就不断产生信息。有了信息，就一直存在对这些信息进行管理的需求，这就催生了数据库。从长远的角度来看，只要有人类，就会产生信息，只要有信息，数据库就会一直存在。所以，我们应该有必要学习数据库。

1.2 学习数据库的境界

学习数据库的境界是，理解数据库的原理，理解 SQL 语句的执行过程，能够写出复杂而高效的 SQL 语句。

1.3 Oracle 霸主地位之——为什么要选择学习 Oracle

2009 年 4 月 20 日，那是一个 IT 业不寻常的日子。那天晚上，数据库和服务器领域处于第一梯队的 Oracle 公司与 Sun 公司宣布合并，涉及金额高达 74 亿美元。这场由 Oracle 主导的收购合并让世人大为震惊。Oracle 由此成为 IT 业内的超级巨无霸。通过整合 Sun 公司的硬件、操作系统（Solaris）、Java、OpenOffice 等，剔除了 Oracle 公司的软肋，延伸了 Oracle 公司的产品线。

在当前的形势下，数据库市场竞争激烈，Oracle、SQL Server、DB2 三足鼎立，其他的小型数据库软件厂商也想过来分一杯羹。Oracle 以其高速、稳定响彻业内，市场份额不用质疑。与 Sun 公司的合并，让 Oracle 数据库的霸主地位不可动摇！因此，我们应该能够找到学习 Oracle 数据库的理由！

1.4 数据库（Database）

本节将简要介绍数据库的基础知识。

1.4.1 何为数据库

数据库（Database，简称 DB）是数据的集合。它按照一定的方式存储数据，并检索和利用数据。目前我们所说的“数据库”的概念已经超出了这个范畴，现在所谓的“数据库”是硬件和软件的结合。

1.4.2 何为关系数据库（RDB）

1970 年，E.F.Codd 先生发表了一篇有关数据库的文章——*A Relational Model of Data for Large Shared Data Banks*，这篇文章主要阐述了关系模型的定义，关系数据库的概念由此诞生。从此，数据库进入一个划时代的发展。首先，IBM 公司根据 E.F.Codd 先生的理论开发了“结构化英语查询语言”（Structured English Query Language，简称 SEQUEL），

可以通过这种语言来访问关系数据库。后来，美国国家标准协会 ANSI 和国际标准化组织 ISO 都对“结构化英语查询语言”进行标准化，使“结构化英语查询语言”成为业界的标准。现在，我们通常都把这种语言叫 SQL（Structured Query Language）。

由于 E.F.Codd 先生的杰出贡献，1981 年，他获得了计算机界的最高奖——图灵奖。

1979 年，Oracle 推出第一个商品化的关系数据库系统 Oracle V2。这个版本的推出并没有引起多大的市场反响。但是，Oracle 公司却开拓了关系数据库商业应用的先河。

1986 年，Oracle 公司推出 Oracle V5，这个版本支持分布式查询。

1988 年，Oracle 公司推出 Oracle V6，该版本具备了联机事务处理的能力。

20 世纪 90 年代以后，由于网络的发展，数据库在商业上的应用并不局限于一个办公间、一栋楼，而是扩展到全世界，Oracle 加强了在分布式环境的集成。

后来，随着硬件的发展，Oracle 把自己的数据库推向了高端应用。Oracle 集群（Real Application Cluster，RAC）及 DataGuard 的出现，为用户提供了最大的性能和数据保护，使商业应用能够实现 7×24 小时的目标（每周 7 天，每天 24 小时不间断运行）。

最近，面向对象（Object-Oriented，OO）的 DBMS 已经成为最为突出的数据库管理系统，并找到了许多适当的应用环境，如多媒体领域。面向对象的 DBMS 适于这样的应用，因为在一个几乎非事务性的环境中，它们具有控制复杂数据类型的能力。由于竞争，RDBMS 厂商为了提供包括文本、音频、图像和视频数据类型的应用，已经制造出了商业可用的通用服务器。

此外，多维数据库（Multi-Dimensional Database，MDD）也分享了部分数据库市场份额，所谓多维数据库，是将数据存放在一个 *n* 维数组中，而不是像关系数据库那样以记录的形式存放。

1.4.3 何为数据库管理系统（DBMS）

数据库管理系统（Databases Management System，DBMS）就是用于管理数据库的**软件**。它负责数据的存储、访问、安全及数据的一致性控制。我们把数据存储到数据库中，或者从数据库中检索数据，都是通过这些软件来实现的。类似的软件有 Oracle、DB2、SQL Server 等。虽然它们都用于数据库的管理，但是它们都拥有各自独立的体系结构。

1.4.4 何为 E-R 图（Entity-Relationship Model）

进行数据库逻辑模型设计的时候，逻辑模型是通过实体-关系图来表示的，这种表示方法叫“E-R 图”。逻辑模型由以下三个元素组成：

- 实体
- 属性
- 关系
 - **实体**，实体是客观存在的事物，它通过一系列的属性进行描述。一个学生是一

个实体，一件产品也是一个实体。实体和实体间要能够相互区分开。实体的概念并不仅仅局限于物体，还可以是事，如手机维修记录就可以看成一个实体。

- **实体集**，所有实体的集合叫实体集，如所有学生的集合就是一个实体集。
- **属性**，属性是实体的组成部分，它描述实体某方面的特性。如果学生是一个实体，那么学生的名字可以看成是这个实体的属性。
- **关系**，关系用于描述两个集合的元素之间的联系，这是数学的范畴。关系模型本来就是建立在数学基础之上的。简单地说，关系就是两个实体之间的联系。两个实体之间有下面几种关系：
 - ✓ 一对一关系，对于实体集 A 中的每个实体，在实体集 B 中只有一个实体与之对应；反之，对于实体集 B 中的每个实体，在实体集 A 中也只有一个实体与之对应。如图 1-1 所示，国家是一个实体，总统也是一个实体，一个国家当前只能有一个总统，一个总统当前也只能在一个国家任职。
 - ✓ 一对多关系，对于实体集 A 中的每个实体，实体集 B 中有多个实体与之对应；反之，对于实体集 B 中的每个实体，实体集 A 中只能有一个实体与之对应。如图 1-2 所示，班主任是一个实体，学生也是一个实体，一个班主任可以对应多个学生，而一个学生只能有一个班主任。

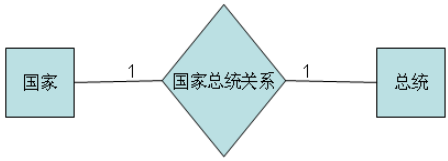


图 1-1　一对一关系



图 1-2　一对多关系

 - ✓ 多对多关系，对于实体集 A 中的实体，实体集 B 中有多个实体与之对应；对于实体集 B 中的实体，实体集 A 中有多个实体与之对应。如医生和病人的关系，一个医生可以给多个病人治病，一个病人也可以接受多个医生的治疗，如图 1-3 所示。

1.4.5　何谓表（Table）

在关系数据库系统中，最基本的元素是表，表又称“关系”，表以二维的形式对数据进行组织。每个表中有 0 行或者多行数据，每一行数据有一列或者多列。图 1-4 是一个学生信息表的例子。

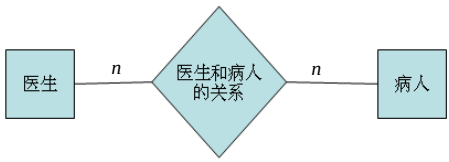


图 1-3　多对多关系

姓名 (Name)	性别 (Sex)	年级 (Grade)	家庭住址 (Address)
王芳	女	一	西城区
李才	男	三	门头沟区
高高	男	一	海淀区

图 1-4　学生信息表

列又称为字段（Field）或者属性，如本例中的姓名（Name）就是一列。列用于描述对象的特性。如本例中，学生是一个对象，学生的姓名是一个属性。

行（Row），又称元组（Tuple）。表中的每行由若干个字段组成，描述一个对象的信息。每个字段描述了该对象的一种属性或性质。如本例中，每一行由学生姓名（Name）、性别（Sex）、年级（Grade）、家庭住址（Address）组成。

1.5 数据库在软件架构中的地位

如图 1-5 所示，在主流的软件应用中，一个应用通常由三层结构组成：展示层、中间层（中间件）、数据库层。展示层负责数据的展示；中间层负责商业逻辑的处理；数据库层用于存放数据。

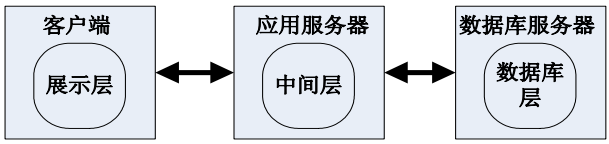


图 1-5 软件架构

1.6 数据库中的数据库——数据字典

数据字典（Data Dictionary）是 Oracle 数据库的核心组成部分之一。它由一系列的表及视图组成，这些表和视图与平常我们所说的表和视图没有多大区别，唯一不同的是，这些表和视图是在创建数据库时由 Oracle 自动创建的，并存放在系统表空间（System Tablespace）中，当数据库启动的时候，系统表空间将自动在线（Online）。这些表和视图的拥有者是用户 sys。这些表和视图由 Oracle 自动维护，我们无权修改，只能以只读（Read Only）的方式对数据字典进行访问。

数据字典存储有关数据库的信息，这些信息包括：

- 所有 Oracle 用户的信息；
- 表空间和数据文件的信息；
- 权限及角色的信息，如哪个用户拥有哪些权限；
- 模式对象的信息（如表、视图、过程、函数、包及触发器等）；
- 完整性约束的信息；
- 有关数据库中对象的空间分布及当前使用情况；
- 审计信息（如谁存取或修改过对象）；
- 其他一般的数据库信息。



1.6.1 数据字典的结构

数据字典由以下两部分组成：

■ 基表（Base Tables）

基表是真正存放有关数据库信息的地方，只有 Oracle 能够读/写这些表，用户不能直接访问这些表。这些表中的数据以加密的形式存在。

■ 视图（User-Accessible Views）

在基表的基础上，Oracle 创建了视图，这些视图是对基表进行加工和汇总形成的。我们不可以直接访问基表，但是我们可以访问这些视图。



1.6.2 数据字典的使用

虽然我们不能修改数据字典，但是，当我们输入 DDL 语句的时候，Oracle 会在后台自动修改数据字典的信息。

如我们输入 SQL 语句：DROP TABLE students（删除表 students），Oracle 会把存放在数据字典中有关 students 表的信息删除。

当数据库打开时，数据字典总是可用，它存放在 System 表空间中。数据字典包含视图集，在许多情况下，每个视图集有三种视图，这些视图包含类似信息，并以 USER、ALL 和 DBA 作为前缀。

- 前缀为 USER 的视图叫用户视图（在当前用户的模式内）。
- 前缀为 ALL 的视图叫扩展用户视图（包括用户有权限访问的内容）。
- 前缀为 DBA 的视图叫 DBA 视图（为全部用户可存取的视图）。



本章总结

本章主要介绍学习数据库的必要性及数据库的基本知识，对于开发人员来说，只需了解关系数据库（RDBMS）的基本知识即可，对于一些学术性很强的概念，不必过分较真！

第 2 章　数据库安装不求人之——自个安装 Oracle 数据库

本章介绍 Oracle 服务器端和客户端的安装，并配有安装视频。安装视频是本书的有益补充！



2.1　Oracle 服务器软件安装之——Linux 平台

本节将讲述 Oracle 11g 在 Linux 平台上的安装。

本例的安装环境是：

数据库版本——Oracle 11g R2

操作系统版本——Red Hat Enterprise Linux 5

安装过程会经历两个大的阶段：操作系统准备和启动 Oracle 的安装。操作系统准备阶段是最重要的一个阶段，它直接影响着安装的成败！



注：安装脚本存放在光盘的\script\install11g_linux.sql 中。



2.1.1　准备操作系统

操作系统准备包括下面几个步骤：

 STEP 01　下载 Oracle 11g R2 的安装文件。

 STEP 02　硬件检查。

 STEP 03　软件检查。

 STEP 04　创建用户和组。

 STEP 05　创建相应的目录。

 STEP 06　修改用户的 SHELL 的限制。

 STEP 07　修改用户的环境配置。

 STEP 08　修改内核参数。



注：除 STEP 07、STEP 01 外，其他步骤都以操作系统用户 root 的身份完成（也就是要以 root 登录服务器）。

1. 下载 Oracle 11g R2 的安装文件

可以从 Oracle 公司的官方网站 http://www.Oracle.com 下载安装文件。11g R2 共由两个安装文件组成：linux_11gR2_database_1of2.zip 与 linux_11gR2_database_2of2.zip。

2. 硬件检查

硬件检查将要检查内存、交换空间、临时目录/tmp 和磁盘空间。

内存检查

执行下面的命令检查服务器内存，Oracle 11g 要求最低 1GB 内存。

```
[root@zzp home]# grep MemTotal /proc/meminfo
MemTotal:     1132672 kB
```

执行下面的命令查看共享内存的大小。

```
[root@zzp home]# df -k /dev/shm/
Filesystem         1K-blocks     Used Available Use% Mounted on
tmpfs                566336        0    566336   0% /dev/shm
```

交换空间检查

执行下面的命令查看交换空间的大小。Oracle 11g 交换空间的要求是：

- 如果物理内存介于 1GB 到 2GB 之间，交换空间应该是物理内存的 1.5 倍。
- 如果物理内存介于 2GB 到 16GB 之间，交换空间应该是物理内存的 1 倍。
- 如果物理内存大于 16GB，交换空间也应该大于 16GB。

```
[root@zzp home]# grep SwapTotal /proc/meminfo
SwapTotal:    2293752 kB
```

执行下面的命令可以查看有效的内存和交换空间。

```
[root@zzp home]# free
             total       used       free     shared    buffers     cached
Mem:       1132672     627704     504968          0      67652     382088
-/+ buffers/cache:     177964     954708
Swap:      2293752          0    2293752
```

临时目录/tmp 检查

执行下面的命令查看临时目录的大小，临时目录的大小至少是 1GB。临时目录和内存通常是越大越好。

```
[root@zzp home]# df -k /tmp
Filesystem         1K-blocks     Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                     11839616   4777532   6450964  43% /
```

磁盘空间检查

执行下面的命令查看磁盘空间的大小，确认是否有足够大的磁盘空间安装 Oracle。

Oracle 对空间的要求，请参考表 2-1 及表 2-2。

```
[root@zzp home]# df -k
Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                     11839616   4777532   6450964  43% /
/dev/hda1              101086     11987     83880  13% /boot
tmpfs                  566336         0    566336   0% /dev/shm
/dev/hdc              2935370   2935370         0 100% /media/RHEL_5.4 i386 DVD
/dev/hdd1               7224512   4728168   2129360  69% /installfile
```

表 2-1　Linux x86-32 空间要求

Oracle 11g	软件占用空间	数据文件占用空间
企业版	3.95GB	1.7GB
标准版	3.88GB	1.5GB

表 2-2　Linux x86-64 空间要求

Oracle 11g	软件占用空间	数据文件占用空间
企业版	4.35GB	1.68GB
标准版	3.73GB	1.48GB

3．软件检查

软件检查将进行操作系统检查、内核检查、包的检查。

操作系统检查

执行下面的命令，得到操作系统的发行版本，本例中的发行版本是 RedHat Enterprise Linux Server release 5.4，而 Oracle 要求的版本是 RedHat Enterprise Linux 5 Update 2（或者更高版本），说明当前操作系统的版本满足 Oracle 的安装要求。

```
[root@zzp home]# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 5.4 (Tikanga)
```

执行下面的命令，还可以获得其他有关操作系统版本的信息。

```
[root@zzp home]# lsb_release -a
LSB   Version:         :core-3.1-ia32:core-3.1-noarch:graphics-3.1-ia32:
graphics-3.1-noarch
Distributor ID: RedHatEnterpriseServer
Description:    Red Hat Enterprise Linux Server release 5.4 (Tikanga)
Release:       5.4
Codename:      Tikanga
```

内核检查

执行下面的命令，检查 Linux 的内核版本，Oracle 要求，对 RedHat Enterprise Linux 5，内核的版本应该是 2.6.18（或者更高），本例中的操作系统内核满足 Oracle 的安装要求。

```
[root@zzp home]# uname -r
2.6.18-164.el5
```

包的检查

在 RedHat Enterprise Linux 5 中，Oracle 11g R2 要求系统安装下面的包（或者该包更高的版本）。

- binutils-2.17.50.0.6
- compat-libstdc++-33-3.2.3
- elfutils-libelf-0.125
- elfutils-libelf-devel-0.125
- gcc-4.1.2
- gcc-c++-4.1.2
- glibc-2.5-24
- glibc-common-2.5
- glibc-devel-2.5
- glibc-headers-2.5
- ksh-20060214
- libaio-0.3.106
- libaio-devel-0.3.106
- libgcc-4.1.2
- libstdc++-4.1.2
- libstdc++-devel-4.1.2
- make-3.81
- sysstat-7.0.2
- unixODBC-2.2.11
- unixODBC-devel-2.2.11

使用命令 rpm –q 查询一个包的存在性。

```
[root@zzp ~]# rpm -q elfutils-libelf-0.125
package elfutils-libelf-0.125 is not installed
```


从输出可以看出 elfutils-libelf-0.125 没有安装在系统中，但是，也许该包的高版本已经被安装在系统中，去掉包的版本号，然后进行查询，如下所示：

```
[root@zzp ~]# rpm -q elfutils-libelf
elfutils-libelf-0.137-3.el5
```

命令执行结果显示，包 elfutils-libelf 的高版本已经安装在系统中。

如果一个包不存在，我们可以从 Linux 操作系统的安装盘中获得该包，也可以从网上进行搜索。获得该包以后，把包上传到服务器上，然后进入包所在的路径，使用命令 rpm –ivh 进行安装，如下所示：

```
[root@zzp installfile]# rpm -ivh sysstat-7.0.2-3.el5.i386.rpm
```

```
    warning: sysstat-7.0.2-3.el5.i386.rpm: Header V3 DSA signature: NOKEY, key
ID 37017186
    Preparing...                ###########################################
[100%]
      1:sysstat                 ###########################################
[100%]
```

4．创建用户和组

Oracle 安装需要在操作中创建 3 个组和 1 个操作系统用户：

📖 Oracle Inventory 组（Oracle Inventory Group）

创建 Oracle Inventory 组 oinstall，oinstall 是组的名字：

```
    [root@zzp home]#/usr/sbin/groupadd oinstall
```

📖 数据库管理员组（Database Administrators Group）

创建数据库管理员组 dba：

```
    [root@zzp home]#/usr/sbin/groupadd dba
```

📖 数据操作员组（Database Operation Group）

创建数据操作员组 oper：

```
    [root@zzp home]#/usr/sbin/groupadd oper
```

📖 操作系统用户

执行下面命令，创建一个操作系统用户，用户名是 oracle，该用户将是 Oracle 软件安装的拥有者。该用户必须属于 Oracle Inventory 组（也就是 oinstall）。

```
    [root@zzp home]#/usr/sbin/useradd -g oinstall -G dba,oper oracle
```

用户 oracle 属于组 oinstall、dba 和 oper。创建操作系统用户 oracle 以后，可以执行命令 id 标识用户的存在性。

```
    [root@zzp home]# id oracle
    uid=501(oracle) gid=501(oinstall) groups=501(oinstall),502(dba),503(oper)
context=root:system_r:unconfined_t:SystemLow-SystemHigh
```

此外，我们还可以使用命令 passwd 修改用户 oracle 的密码。

```
    [root@zzp home]# passwd oracle
    Changing password for user oracle.
    New UNIX password:
    BAD PASSWORD: it does not contain enough DIFFERENT characters
    Retype new UNIX password:
    passwd: all authentication tokens updated successfully.
```

5．创建相应的目录

Oracle 需要创建 5 个目录：

🕮 Oracle 根目录（基目录）

创建 Oracle 根目录/myoracle/app：

```
[root@zzp home]# mkdir -p /myoracle/app
```

修改目录的属主：

```
[root@zzp home]# chown -R oracle:oinstall /myoracle/app
```

修改目录的读、写、执行权限：

```
[root@zzp home]# chmod -R 775 /myoracle/app/
```

🕮 Oracle 主目录

创建 Oracle 主目录/myoracle/app/product/11.2.0/db_1：

```
[root@zzp home]# mkdir -p /myoracle/app/product/11.2.0/db_1
```

修改目录的属主：

```
[root@zzp home]# chown oracle:oinstall  /myoracle/app/product/11.2.0/db_1
```

修改目录的读、写、执行权限：

```
[root@zzp home]# chmod 775 /myoracle/app/product/11.2.0/db_1
```

🕮 Inventory 目录

创建 Inventory 目录/myoracle/oraInventory：

```
[root@zzp home]# mkdir -p /myoracle/oraInventory
```

修改目录的属主：

```
[root@zzp home]# chown oracle:oinstall  /myoracle/oraInventory
```

修改目录的读、写、执行权限：

```
[root@zzp home]# chmod 775 /myoracle/oraInventory
```

🕮 数据文件目录

创建数据文件目录/myoracle/oradata：

```
[root@zzp home]# mkdir /myoracle/oradata
```

修改目录的属主：

```
[root@zzp home]# chown oracle:oinstall /myoracle/oradata
```

修改目录的读、写、执行权限：

```
[root@zzp home]# chmod 775 /myoracle/oradata
```

🕮 闪回区目录

创建闪回区目录/myoracle/oradata：

```
[root@zzp home]# mkdir /myoracle/flash_recovery_area
```

修改目录的属主：

```
[root@zzp home]# chown oracle:oinstall /myoracle/flash_recovery_area
```

修改目录的读、写、执行权限：

```
[root@zzp home]# chmod 775 /myoracle/flash_recovery_area
```

6．修改用户的 SHELL 的限制

为了提高性能，你需要为用户 oracle 提高下面 SHELL 的限制。
编辑文件 limits.conf：

```
[root@zzp ~]#vi /etc/security/limits.conf
```

在文件中加入（如果条目不存在）或者修改下面的内容。

```
oracle soft nproc 2047
oracle hard nproc 16384
oracle soft nofile 1024
oracle hard nofile 65536
```

编辑文件 login：

```
[root@zzp ~]#vi /etc/pam.d/login
```

在文件中加入（如果条目不存在）或者修改下面的内容。

```
session required /lib/security/pam_limits.so
session required pam_limits.so
```

编辑文件 profile：

```
[root@zzp ~]#vi /etc/profile
```

在文件中加入（如果条目不存在）或者修改下面的内容。

```
if [ $USER = "oracle" ]; then
if [ $SHELL = "/bin/ksh" ]; then
ulimit -p 16384
ulimit -n 65536
else
ulimit -u 16384 -n 65536
fi
fi
```

7．修改用户的环境配置



注：修改用户 oracle 的环境变量需要使用用户 oracle 的身份登录操作系统。

STEP 01 以用户 oracle 的身份登录操作系统。

```
[root@zzp /]# su - oracle
```

STEP 02 查看当前目录是否在操作系统用户 oracle 的主目录。

```
[oracle@zzp ~]$ pwd
/home/oracle
```

STEP 03 找到环境配置文件.bash_profile。

```
[oracle@zzp ~]$ ls -a
.            .bash_profile  .gconfd         .mozilla       .Xauthority
..           .bashrc        .gnome2         oradiag_oracle  .zshrc
.bash_history  .emacs        .gnome2_private  .viminfo
.bash_logout   .gconf        .kde            .vnc
```

STEP 04 修改环境配置文件。

```
[oracle@zzp ~]$ vi .bash_profile
```

在该文件中加入下面的内容：

```
export ORACLE_BASE=/myoracle/app
export ORACLE_HOME=/myoracle/app/product/11.2.0/db_1
export ORACLE_SID=zdb
export umask=022
export PATH=$PATH:$ORACLE_HOME/bin
```

其中，ORACLE_BASE 是 Oracle 的根目录。ORACLE_HOME 是 Oracle 的主目录。ORACLE_SID 是实例名（如果要创建数据库）。PATH 设置命令的搜索路径。

可以使用下面的命令查看环境变量。

```
[oracle@zzp ~]$env
```

8. 修改内核参数

修改内核参数配置文件 sysctl.conf：

```
[root@zzp ~]vi /etc/sysctl.conf
```

在该文件中加入如下的内容，下面列出的只是最小值，如果要修改这些内核参数的值，我们输入的值一定要大于或者等于下面的值。

```
fs.aio-max-nr = 1048576
fs.file-max = 6815744
kernel.shmall = 2097152
kernel.shmmax = 536870912
kernel.shmmni = 4096
kernel.sem = 250 32000 100 128
net.ipv4.ip_local_port_range = 9000 65500
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
```

```
net.core.wmem_default = 262144
net.core.wmem_max = 1048586
```

为了使修改的内核参数马上生效，应执行下面的命令：

```
[root@zzp /]#/sbin/sysctl -p
```

可以使用类似于下面的命令查看内核参数的值：

```
[root@zzp /]#/sbin/sysctl -a | grep kernel.shmmax
```



2.1.2 启动 Oracle 安装

在操作系统准备完成以后，可以按照下面的步骤启动 Oracle 的安装。

STEP 01 把安装文件上传到服务器上（本例中服务器的路径是/installfile/）。

STEP 02 解压。

ⓐ 进入到压缩文件的路径。

```
[oracle@zzp ~]$cd /installfile/
```

ⓑ 解压第一个文件。

```
[oracle@zzp ~]$unzip linux_11gR2_database_1of2.zip
```

ⓒ 解压第二个文件。

```
[oracle@zzp ~]$unzip linux_11gR2_database_2of2.zip
```

STEP 03 从客户机使用 X Window 工具登录 Linux 服务器（本例中使用 VNC）。

STEP 04 使用 VNC 登录以后，单击鼠标右键，打开一终端（也就打开一命令行窗口）。

ⓐ 禁用访问控制（这和能否启动图形界面有很大关系）。

```
[root@zzp ~]# xhost +
access control disabled, clients can connect from any host
```

xhost 命令用于添加或删除 X 服务器接受连接的机器列表上的主机名。

ⓑ 切换到用户 oracle。

```
[root@zzp ~]# su - oracle
```

ⓒ 进入解压路径，子目录 database 是解压以后生成的目录。

```
[oracle@zzp ~]$ cd /installfile/database/
```

ⓓ 找到安装文件 runInstaller。

```
[oracle@zzp database]$ ls
doc  install  response  rpm  runInstaller  sshsetup  stage welcome.html
```

ⓔ 启动 Oracle 的安装界面。

```
[oracle@zzp database]$ ./runInstaller
Starting Oracle Universal Installer...
```

进入 Oracle 的安装界面以后，请按照提示一步步进行安装，这里启动以后的图形界面和在 Windows 平台启动的图形界面类似，我们就不再重复叙述。

需要注意的是，在安装的过程中，图形界面会提示要以 root 的身份运行脚本 root.sh，这时候，我们离开安装界面，以 root 身份打开一命令行窗口，然后执行脚本 root.sh，如下所示：

```
[root@zzp ~]# /myoracle/app/product/11.2.0/db_1/root.sh
Running Oracle 11g root.sh script...
The following environment variables are set as:
    ORACLE_OWNER= oracle
    ORACLE_HOME=  /myoracle/app/product/11.2.0/db_1
```

如果你想了解更多 Oracle 11g 在 Linux 上的安装细节，请参考本书附带的关盘。

2.2 Oracle 服务器软件安装之——Windows 平台

本书将重点介绍 Oracle 11g 在 Windows XP 平台的安装，并在安装的过程中引入安装的一些基本概念，**这是本书的一大特点。**

本书用到的软件如下：

数据库软件——Oracle Database 11g Release 1 (11.1.0.6.0) for Microsoft Windows (32-bit)

操作系统——Windows XP

2.2.1 Oracle 服务器软件安装

本节将介绍 Oracle 服务器软件的安装。安装服务器软件时，内存和磁盘空间最好能满足 Oracle 的最低要求。在安装涉及的路径中，不要包含中文字符。

1．Oracle Universal Installer 介绍

Oracle 统一安装器（Oracle Universal Installer）是一个图形界面工具，用于安装和卸载 Oracle 软件。安装和卸载 Oracle 的许多软件都会用到该工具。该工具为用户提供了统一的界面。

2．Oracle 安装软件的获得

我们可以从 Oracle 的官方网站免费下载 Oracle 数据库安装软件，但是，这样的软件只能用于学习和研究，禁止用于商业用途。这样下载的软件和付费后的软件在功能上没有什么区别！

Oracle Database 11g Release 1 (11.1.0.6.0) for Microsoft Windows (32-bit)的下载地址是：

http://www.oracle.com/technology/software/products/database/oracle11g/111060_win32soft.html。



注意：下载前，需要选中 ○ **Accept** License Agreement。

3．Oracle 安装前期的准备

Oracle 安装的前期准备工作尤为重要，许多莫名其妙的错误是由于前期的准备工作引起的。安装 Oracle 服务器软件时，需要满足下面的要求，如表 2-3 及表 2-4 所示。

表 2-3　硬件配置（Windows 32 位操作系统）要求

硬　　件	要　　求
物理内存	最低 1GB
虚拟内存	最低是内存的 2 倍，多多益善
磁盘空间	5GB 以上
处理器（CPU）	550 MHz 以上
显卡	256 色

表 2-4　软件配置（Windows 32 位操作系统）要求

软　　件	要　　求
系统架构	处理器：Intel (x86)，AMD64，Intel EM64T
操作系统	■Windows 2000 Service Pack 1 以上 ■Windows Server 2003 ■Windows XP Professional ■Windows Vista
网络协议	■TCP/IP ■TCP/IP with SSL ■Named Pipes

4．配置检查

在安装之前，需要对内存和虚拟内存进行检查。如果不满足要求，则需要设法满足。有时候，即使不满足条件，Oracle 也能安装成功，但是，建议不要在生产系统中进行这样的尝试。

➢ 内存检查

进入 Windows XP，单击“开始”→“控制面板”→“性能和维护”→“系统”，将会看到如下所示的内存信息。

```
Celeron(R) CPU 2.93GHz
2.93GHz,1.99GB 的内存
物理地址扩展
```

➢ 虚拟内存的检查

进入 Windows XP，单击“开始”→“控制面板”→“性能和维护”→“系统”→“高级”→“设置”→“高级”，将会列出虚拟内存的信息。在此亦可对虚拟内存进行修改。

5．Oracle 安装进行中

好了，一切准备就绪，让我们开始 Oracle 软件的安装吧！

STEP 01 以管理员的身份登录操作系统（安装需要管理员权限）。

STEP 02 找到安装文件 win32_11gR1_database_1013.zip，并进行解压。

STEP 03 双击“database\install\oui.exe”，启动 Oracle 11g 的安装，弹出如图 2-1 所示的界面。

单击 下一步(N) 按钮，出现如图 2-2 所示的界面。

STEP 04 选择安装类型。



图 2-1 安装首页



图 2-2 选择安装类型



知识点索引——安装类型

- ✓ 企业版安装（Enterprise Edition），安装许可的数据库选项、数据库配置和管理工具，还包括所有标准版的安装。企业版安装的选项比较全面，一般我们选择企业版安装（Enterprise Edition）。
- ✓ 标准版安装（Standard Edition），安装集成的管理工具、分布式组件、复制和 Web 特性。标准版安装可用的 Oracle 特性（组件）比企业版少。如果从空间的角度考虑，我们可以选择标准版安装。标准版安装（Standard Edition）适合一个工作组、一个部门或者一家小公司。
- ✓ 个人版安装（Personal Edition），也可以说是限制最大的安装，这种安装方法只安装企业版的某些特性，且只支持单用户。个人版安装还不支持集群（Oracle Real Application Clusters，RAC）。由于单用户的限制，这种安装只适合学习。
- ✓ 定制安装（Custom），给用户提供一个界面，让用户选择需要安装的组件。从节省磁盘空间的角度考虑，我们可以选择定制安装。安装需要的组件，不安装多余的组件。

通常选择“企业版”，单击 下一步(N) 按钮，出现如图 2-3 所示的界面。

STEP 05 设置目录。



图 2-3　设置目录



知识点索引——基目录和主目录

Oracle 基目录（Oracle Base Directory）

Oracle 基目录(又叫 Oracle 的根目录)就是 Oracle 的**顶级目录。**第一次安装 Oracle 的时候，Oracle Universal Installer 会提示创建这个目录。下次再安装 Oracle 其他组件的时候，Oracle Universal Installer 会自动检测到 Oracle 的基目录。

Oracle 主目录（Oracle Home Directory）

Oracle 主目录就是 Oracle 软件的存放目录。安装数据库时，Oracle 软件会安装到该目录下。**该目录必须是 Oracle 基目录 (Oracle Base) 的子目录。**

基目录和主目录可以保留默认设置，也可以根据自己的需要进行调整，输入基目录和主目录后，单击 下一步(N) 按钮，出现如图 2-4 所示的界面。



先决条件检查。



图 2-4　先决条件检查

知识点索引——先决条件检查

所谓“先决条件检查”，即是指检查硬件（如内存、磁盘空间）及软件等是否满足 Oracle 的最低安装要求。

如果先决条件未执行，会显示信息“未执行”，单击 未执行，使其变成 用户已验证，表示用户已经手工验证了该项的正确性；如果先决条件检查通过，会显示“成功”；如果先决条件检查未通过，会显示其失败原因。

单击 未执行，单击 下一步(N) 按钮，出现如图 2-5 所示的界面。

STEP 07 选择配置选项。

选择“仅安装软件”，单击 下一步(N) 按钮，出现如图 2-6 所示的界面。

STEP 08 安装概要。



图 2-5　选择配置选项



图 2-6　安装概要

安装“概要”列出了安装源文件的路径、目标路径、产品支持的语言、所需磁盘空间、要安装的组件等信息。

单击 安装(I) 按钮，安装开始，进入如图 2-7 所示的界面。

STEP 09 安装进行中。



图 2-7　安装进行中

该步骤耗时较长，请耐心等待。

STEP 10 安装成功，界面如图 2-8 所示。



图 2-8　安装成功

长时间的等待之后，终于迎来 Oracle 软件的安装成功！



知识点索引—安装 Oracle 软件和创建数据库

“安装 Oracle 软件”和“创建数据库”是两个过程。到目前为止，我们仅仅完成了 Oracle 软件的安装，并未创建任何数据库。

单击 退出(X) 按钮并确认，退出 Oracle 服务器软件的安装。



2.2.2 Oracle 服务器软件安装之——故障排查

由于 Oracle 体系庞大，涉及硬件、软件（尤其是操作系统），因此，在安装的过程中，遭遇错误在所难免。如何解决这些错误呢？

如果遇到错误，我们首先应该重新检查一下软件和硬件，看看是否都满足 Oracle 的最低安装要求。这些检查包括内存、虚拟内存、硬盘、网络、操作系统及各种补丁包等。

在确认 Oracle 的最小安装要求满足后，再检查 Oracle 安装日志文件。安装 Oracle 的时候，Oracle 会把安装的详细过程记录到一些文件中，这些文件就是“**安装日志文件**”。这些文件可以帮助我们快速定位安装过程中遇到的疑难问题，使我们能够顺利进行安装，因此，我们必须找到这些日志文件。

■ Windows 平台安装日志文件的查找

Windows 中的安装日志文件存放在 SYSTEM_DRIVE:\Program Files\Oracle\Inventory\logs 下，不需要进行额外的查找。SYSTEM_DRIVE 表示 Windows 系统所在的盘符，一般是 C 盘。

■ UNIX 平台安装日志文件的查找

STEP 01 找到文件 oraInst.loc。

oraInst.loc 文件的位置如下：

- 对于 AIX、Linux x86 或 Linux Itanium，文件的位置是/etc/oraInst.loc；
- 对于其他 UNIX 操作系统，文件的位置是/var/opt/Oracle/oraInst.loc。

从文件 oraInst.loc 中，找到 Inventory 的位置。

```
$ cat /var/opt/Oracle/oraInst.loc
inventory_loc=/Oracle/app/Oracle/oraInventory
inst_group=oinstall
```

这里的 Inventory 目录是/Oracle/app/Oracle/oraInventory。

STEP 02 进入 Inventory 目录，找到安装日志文件。

```
$ cd /Oracle/app/Oracle/oraInventory
$ cd logs
```

STEP 03 查看安装日志文件的内容。

安装日志文件的文件名的格式通常是 installActionsdate_time.log。

■ 查看操作系统日志文件

除了 Oracle 产生的安装日志文件，操作系统也会产生日志文件，这种类型的文件，被我们叫做“操作系统日志文件”。“操作系统日志文件”记录操作系统级的错误或者操作系统的重大事件。这些文件的内容有助于我们排除 Oracle 的故障。因此，我们应该学会查找和使用这些文件。

2.2.3 创建 Oracle 11g 数据库（Windows 平台）

在前面的章节中，我们只是介绍了 Oracle 数据库服务器的安装，并未涉及数据库的创建。也就是说，到目前为止，我们还没有一个自己的数据库，相信大家期盼创建数据库这一刻已经许久了吧。

有两种方式可以创建数据库：一种是使用图形界面工具 DBCA（Database Configuration Assistant），又叫“数据库配置助手”，该工具允许与用户交互，界面友好，是使用频率最高的工具；另外一种方式是使用语句 CREATE DATABASE，这是一种基于命令行的方式。两种方式创建数据库的效果几乎相同。由于 DBCA 简单、易于掌握，本书将重点介绍 DBCA，在创建数据库的过程中，将引入 Oracle 体系结构的一些基本概念，这是本书的特点，希望大家能喜欢这种方式。

STEP 01 创建目录。

在操作系统中，创建目录 F:\newdb，该目录用于存放 Oracle 所有的数据文件。

STEP 02 启动 DBCA。

ⓐ 以操作系统管理员的身份登录操作系统。

ⓑ 依次选择“开始（Start）”→“所有程序（All Programs）”→“Oracle 主目录（HOME_NAME）”→“配置和移植工具（Configuration and Migration Tools）”→“数据库配置助手（Database Configuration Assistant）”，即可启动 DBCA。

如果在 UNIX、Linux 平台，输入命令 dbca 启动 DBCA。dbca 命令位于 Oracle_home/bin 下。启动 DBCA 后，其界面如图 2-9 所示。

知识点索引——在 Linux 下启动 DBCA

在 Linux 平台，执行下面的过程启动 DBCA：

1）切换到用户 oracle。

```
[root@zzp ~]# su - oracle
```

2）输入命令 dbca。

```
[oracle@zzp ~]$dbca
```

STEP 03 DBCA 首页。

单击 下一步(N) 按钮，进入如图 2-10 所示的界面。

STEP 04 DBCA 配置选择。



图 2-9　DBCA 首页



图 2-10　DBCA 配置选择

选择“创建数据库”，单击 下一步(N) 按钮，进入如图 2-11 所示的界面。

STEP 05 数据库模板。



图 2-11　数据库模板

我们可以创建三种类型的数据库：

- 一般用途或事务处理（General purpose or transaction processing）
- 数据仓库（Data warehouse），也可以叫决策支持系统（Decision Support Systems，DSS）。在这种类型的数据库中，用户执行大量的联机分析处理（Online Analytical Processing，OLAP）。联机分析处理将执行大量的查询（Query），涉及大量的数据。这是一种以“读”为主的数据库。
- 定制数据库（Custom Database），顾名思义，就是可以根据需要对数据库进行定制。

选择“定制数据库”，单击 下一步(N) 按钮，进入如图 2-12 所示的界面。

STEP 06 设置数据库名和实例名。



图 2-12　数据库名和实例名

知识点索引——数据库名和实例名

数据库名（Database Name）

数据库名用于标识一个数据库。

实例名（System Identifier，SID）

实例名又叫系统标识符（SID），用于唯一地标识一个**实例**，实例是**内存**和**后台进程**的集合。

数据库名和实例名的关系

数据库名（Database Name）和实例名（SID）是两个概念，在 RAC（集群）环境下，多个实例可以同时打开一个数据库。数据库名和实例名可以不同。通常，在单机环境下，数据库名和实例名相同。

本例中，输入全局数据名 zdb 和实例名 zdb 以后，单击 下一步(N) 按钮，进入如图 2-13 所示的界面。

STEP 07 配置企业管理器。

知识点索引——为什么配置企业管理器

如果想通过 Web 页面管理数据库，请选中“配置 Enterprise Manager”。“启用预警通知”是一项比较有用的功能。输入邮箱及 SMTP 的地址，当 Oracle 有异常出现的时候，会通过 E-mail 的形式通知你，说你的数据库哪里出了问题，需要你关注一下。

单击 下一步(N) 按钮，进入如图 2-14 所示的界面。

STEP 08 设置管理员密码。

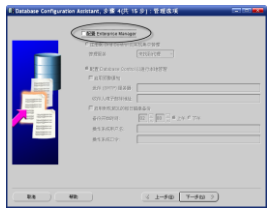


图 2-13 配置企业管理器



图 2-14 管理员密码设置

本例中，把数据库管理员 SYS 和 SYSTEM 的密码都设置成 zero，单击 下一步(N) 按钮，进入如图 2-15 所示的界面。

STEP 09 存储方式选择。



图 2-15 存储方式选择

知识点索引——存储知识

Oracle 提供三种方式对数据进行存储：文件系统（File System）、自动存储管理

（Automatic Storage Management，ASM）和裸设备（Raw Device）。

文件系统（File System）：文件系统是使用得最频繁的一种存储方式。如果使用文件系统，Oracle 将把数据文件存放到文件系统的一个目录中。

- ✓ 普通文件系统，这是在普通磁盘上创建的文件系统。在分布数据文件的时候，建议大家遵循 OFA（Optimal Flexible Architecture）的原则。
- ✓ 建立在逻辑卷（Logical Volume Manager，LVM）或者 RAID 设备上的文件系统。如果要使用这样的文件系统，Oracle 推荐使用先条块化、后镜像（Stripe-And-Mirror）的机制，使用这种机制在提高性能的同时，还可以提高数据的可靠性。

自动存储管理（Automatic Storage Management，ASM）：在这里，我们把它叫做“自动存储管理”。ASM 是一种 Oracle 存储的解决机制，它免去了 DBA 对磁盘 I/O 的手动调整，简化了数据库的动态管理，如：创建和分布数据库、磁盘空间管理。ASM 是 Oracle 10g 的新特性之一。

裸设备（Raw Device）：裸设备也是应用得比较广泛的一种存储技术。它绕过了操作系统，由 Oracle 直接对存放在裸磁盘上的数据进行读/写。对于文件系统来说，如果我们要写数据，Oracle 首先把数据交给操作系统，再由操作系统写磁盘。裸设备的读写机制正好绕过了操作系统这一层。

Oracle 三种存储机制各有千秋，不能一概而论。在对系统进行设计的时候，我们应该进行测试（各个厂商提供的设备不尽相同），系统的部署是建立在测试基础之上的。只有测试以后，我们才能选择使用何种机制来分布 Oracle 的数据。

选择“文件系统”，单击 下一步(N) 按钮，进入如图 2-16 所示的界面。

STEP 10 指定数据文件的分布方式。



图 2-16　数据文件的分布方式

知识点索引——数据文件的分布方式

- ✓ 选择“使用模板中的数据库文件位置（Use Database File Locations from Template）”，则使用 Oracle 预先设置好的模板进行文件的分布。
- ✓ 选择“所有数据库文件使用公共位置（Use Common Location for All Database Files）”，则所有的数据文件都放在相同的目录之下。

✓ **选择“使用 Oracle-Managed Files”，则使用 Oracle 管理的文件，这种方法可以简化文件的管理。例如，当删除一个表空间的时候，Oracle 将自动删除该表空间相应的数据文件（操作系统文件），而不需要 DBA 手工使用操作系统命令删除这些文件。**

在本例中，选择“所有数据库文件使用公共位置”，输入 F:\newdb，单击 下一步(N) 按钮，进入如图 2-17 所示的界面。

STEP 11 配置恢复区和归档。

- 指定快速恢复区，这是 Oracle 10g 的新特性，这个特性可以把数据库闪回（Flash）到以前的某个状态。
- 启用归档，则使创建的数据库运行于归档模式。“启用归档”涉及数据库的备份和恢复，是数据库备份/恢复机制的组成部分。有关归档的相关知识，**请参考本书 22.6.1 节的内容。**

选中“指定快速恢复区”，指定快速恢复区的路径和大小。启用归档，单击 下一步(N) 按钮，进入如图 2-18 所示的界面。

STEP 12 配置数据库组件。

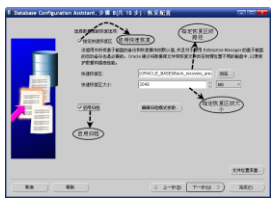


图 2-17　配置恢复区和归档



图 2-18　配置数据库组件

选择安装的数据库组件及这些组件存放的表空间（表空间的概念，**请参考本书第 7 章的内容**）。单击 下一步(N) 按钮，进入如图 2-19 所示的界面。

STEP 13 内存设置。

有两种方式设置内存的大小，一种是“典型”，另外一种是“定制”。“定制”允许用户分别对 SGA（系统全局区）和 PGA（程序全局区）进行单独设置。

初始化参数 SGA_TARGET 控制是否启用内存自动管理。

SGA（系统全局区）和 PGA 的介绍，**请参考本书第 9 章的内容。**

单击 调整大小 标签，出现如图 2-20 所示的界面。

STEP 14 块和进程数的设置。

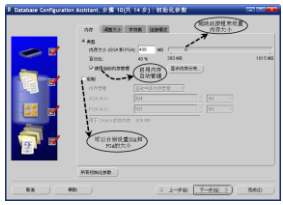


图 2-19　内存设置

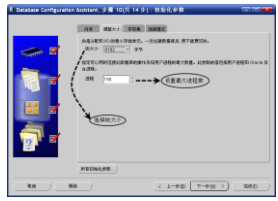


图 2-20　块和进程数的设置

知识点索引——块（Block）

数据块，本书简称为“块”，是 Oracle 最小的存储单位，Oracle 数据存放在“块”中。一个块占用一定的磁盘空间。特别注意的是，这里的“块”是 Oracle 的“数据块”，不是操作系统的“块”。

Oracle 每次请求数据的时候，都是以块为单位。也就是说，Oracle 每次请求的数据是块的整数倍。如果 Oracle 请求的数据量不到一块，Oracle 也会读取整个块。所以说，“块”是 Oracle 读写数据的最小单位或者最基本的单位。

块的标准大小由初始化参数 DB_BLOCK_SIZE 指定。具有标准大小的块称为标准块（Standard Block）。块大小和标准块大小不同的块叫非标准块（Nonstandard Block）。在同一数据库中，Oracle 9i 及以上版本支持同一数据库中同时使用标准块和非标准块。Oracle 允许指定 5 种非标准块。

操作系统每次执行读写（I/O）的时候，是以操作系统的块为单位；Oracle 每次执行读写（I/O）的时候，都是以 Oracle 的块为单位。

如果是数据仓库，块越大越好。

在数据库创建完成以后，就不能够对块的尺寸进行修改。

把进程数设置成 200，其他选项保留默认设置，单击 字符集 标签，进入如图 2-21 所示的界面。

STEP 15 字符集设置。

通常的简体中文字符集是 ZHS16GBK，国家字符集选择 AL16UTF16。 国家字符集允许在一个数据库中存放多国语言表示的数据。

单击 连接模式 标签，进入如图 2-22 所示的界面。

STEP 16 配置共享服务器模式。



图 2-21 字符集设置



图 2-22 配置共享服务器模式

知识点索引——专有服务器模式

专有服务器模式（Dedicated Server Mode）是 Oracle 一种特殊的进程结构。其体系结构如图 2-23 所示。

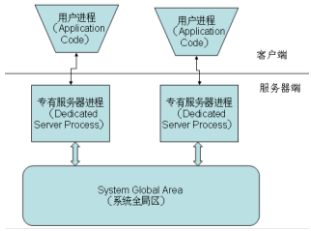


图 2-23　专有服务器模式的体系结构

在专有服务器模式中，用户进程运行在客户端的机器上，专有服务器进程运行在数据库服务器端的机器上。用户进程和专有服务器进程是两种不同类型的进程。当客户端发出连接数据库请求的时候，监听器把客户端（用户进程）连接到专有服务器进程。即使这个服务器进程很空闲，这个专有服务器进程也只为这个用户进程服务。用户进程和专有服务器进程之间是一一对应的关系。

专有服务器模式有自己的优点，它适合于下面的应用：

- ✓ 批处理，批处理使服务器进程一直处于忙碌的状态，这就不会造成服务器进程的空闲，导致系统资源的浪费。
- ✓ 使用 Recovery Manager（RMAN）进行备份或者恢复。

当第一个用户登录的时候，监听器会启动一个专有服务器进程为这个用户服务；当第 *N* 个用户登录的时候，监听器会启动第 *N* 个专有服务器进程为这个用户服务。随着登录用户数量的增加，系统的负担也相应增加。到了一定的时候，数据库服务器将不堪重负，所以，专有服务器模式限制了应用的可扩展性。

有关进程的信息，**请参考本书第 10 章的内容**。



知识点索引——共享服务器模式（Shared Server Mode）

共享服务器模式中，调度进程（Dispatcher）接受多个客户端（用户进程）的连接请求，并把这些请求放到一个公共队列中。空闲的服务器进程从这个公共队列中抓出这些请求，然后完成请求的任务，并把结果放到响应队列中，调度进程再从响应队列中把完成的结果返回给用户进程（客户端）。

由于一个服务器进程不再专为一个用户进程服务，而是为所有的用户进程服务，因此，共享服务器模式在不消耗太多系统资源的情况下，能够支持更多的并发用户连接，提高应用程序的可扩展性。

一个共享服务器模式的系统由下面三个部分组成。

- ✓ 监听器
- ✓ 一个或者多个调度进程（Dispatcher Processes）

✓ 一个或者多个共享服务器进程（Shared Server Processes）

共享服务器的框架结构如图 2-24 所示。

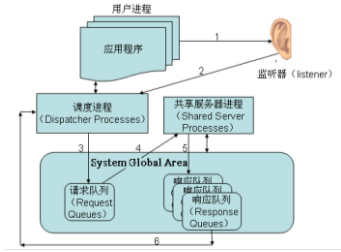


图 2-24　共享服务器模式的体系结构

选择“共享服务器模式”，输入初始创建的服务器进程数为 5，单击 下一步(N) 按钮，进入如图 2-25 所示的界面。

STEP 17　安全设置。

选择“保留增强的 11g 默认安全设置（建议）”，单击 下一步(N) 按钮，进入如图 2-26 所示的界面。

STEP 18　启用自动维护任务。



图 2-25　安全设置

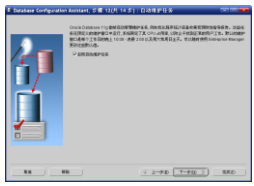


图 2-26　自动维护任务

知识点索引——自动维护任务

自动维护任务（Automatic Maintenance Tasks）指 Oracle 会自动地对数据库进行维护。这些维护包括：自动收集统计信息、自动地对段（Segment）进行监控、自动标识效率低下的 SQL 语句。这是 11g 的新特性。

选择“启用自动维护任务”，单击 下一步(N) 按钮。

STEP 19　数据库存储设置。

单击 控制文件，出现如图 2-27 所示的界面，在该界面中可以修改控制文件的名字和路径，还可以添加新的控制文件。本例中，增加控制文件 mycontrol.ctl，路径是 F:\newdb。

展开 表空间，出现如图 2-28 所示的界面。



图 2-27　增加控制文件

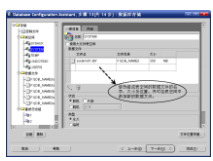


图 2-28　设置表空间

在该界面中，可以修改组成表空间的数据文件的名字、位置及尺寸等参数。

展开 数据文件，出现如图 2-29 所示的界面。

在该界面中，可以修改数据文件的名字、路径和尺寸，还可以添加新的数据文件。

展开 重做日志组，出现如图 2-30 所示的界面。



图 2-29　修改数据文件



图 2-30　修改重做日志组文件

在默认情况下，有三组重做日志组，每组中只有一个成员（文件），但每组中至少需要两个成员。本例中，往重做日志组 1 中添加文件 F:\newdb\myredo02.log。

单击 下一步(N) 按钮，进入如图 2-31 所示的界面。



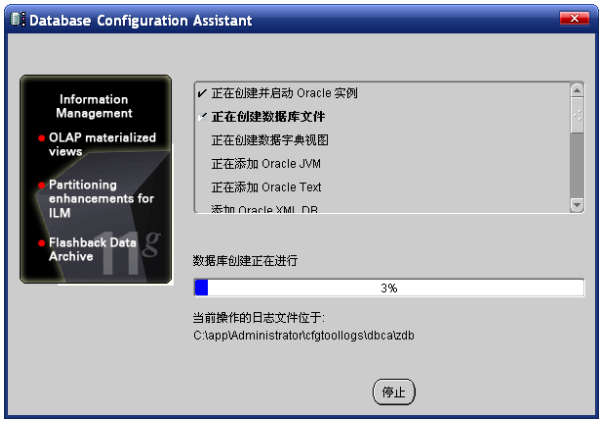
图 2-31　创建选项

单击 完成(F) 按钮并确定，出现如图 2-32 所示的界面，开始创建数据库。

STEP 20　正在创建数据库。

创建数据库的过程较为漫长，需要我们耐心等待。安装完成后，出现如图 2-33 所示的界面。

STEP 21　安装完成。



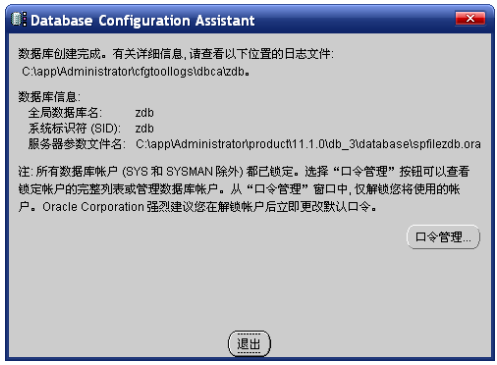


图 2-32　开始创建数据库　　图 2-33　安装完成

我们可以单击 口令管理... 按钮设置 Oracle 内置用户的口令。单击 退出 按钮，Oracle 数据库创建成功完成。至此，我们创建了一个名叫 zdb 的数据库，实例名（SID）也是 zdb。配置监听器和 Oracle 客户端时，将会用到数据库名和实例名，这些信息就是从这里产生的。

2.2.4　Oracle 安装后期——测试安装是否成功

数据库创建以后，我们尚不知创建是否真的成功。有几种方法判断数据库创建成功与否。我们在此介绍最简单的一种判断方法。

STEP 01　打开命令行窗口。

通常单击“启动”→“运行”，输入“cmd”命令，单击“确定”按钮启动命令行窗口。

STEP 02　启动 SQL*Plus，输入用户名和密码，登录数据库。

```
C:\Documents and Settings\Administrator>sqlplus
SQL*Plus: Release 11.1.0.6.0 - Production on 星期一 7 月 20 12:57:53 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
请输入用户名:  sys as sysdba
输入口令:
连接到:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

本例中，用户 sys 的密码是 zero。

STEP 03　在数据库中，执行下列 SQL 语句。

```
SQL> CREATE TABLE b (a int);
表已创建。
SQL> INSERT INTO b VALUES(1);
已创建 1 行。
```

```
SQL> COMMIT;
提交完成。
```

```
SQL> SELECT * FROM b;
```

SQL 语句的执行结果是：

```
         A
--------------------
         1
SQL>
```

本例中创建表 b，往表 b 中成功地插入数据，并显示出已插入的数据。由此，可以判断数据库创建成功。



2.3 Oracle 客户端安装

本节只是简单介绍 Oracle 11g 客户端在 Windows XP 上的安装，有关详细的信息，敬请浏览随书光盘。



STEP 01 解压安装文件 win32_11gR1_client.zip。

一定要把文件全部解压出来才能安装，否则就会出现错误“OUI-10133:登台区无效。此登台区没有可用于安装的 Windows NT，Windows 2000 顶级组件”。



STEP 02 双击“setup.exe”，启动客户端安装。

进入“欢迎使用”界面，单击“下一步”按钮，进入“选择安装类型”界面。



STEP 03 选择安装类型。

在该界面中，通常选择“管理员”，就能满足我们大部分的要求。单击“下一步”按钮，进入“指定主目录详细信息”界面。



STEP 04 指定主目录详细信息。

在该界面中，指定 Oracle 基目录、Oracle 主目录的名字及位置。Oracle 客户端软件将被安装在 Oracle 主目录下。这里的“基目录”、“主目录”和安装数据库时涉及的“基目录”、“主目录”是不同的，不能混淆。本例中基目录是 C:\ora_client，主目录的名字是 OraClient11g_home1，主目录的路径是 C:\ora_client\product\11.1.0\client_2。单击“下一步”按钮，进入“产品特定的先决条件检查”界面。



STEP 05 产品特定的先决条件检查。

在该界面中，Oracle 会检查最低安装要求，如果用户提供的条件不满足 Oracle 的最低安装要求，用户需要对安装条件进行调整。如果检查全部通过，单击“下一步”按钮，进入“概要”界面。



STEP 06 概要。

“概要”会列出所有的安装信息，包括安装路径、要被安装的组件等。

单击“安装”按钮，客户端的安装正式开始。接下来的过程很简单，无须进行详细描述了。

阅读参考：学完本章以后，接下来，请先阅读本书第 4 章的内容。



本章总结

本章介绍 Oracle 数据库服务器端和客户端的安装。客户端的安装比较简单，服务器端的安装就比较复杂一些。Oracle 数据库服务器端安装成功的关键是要满足 Oracle 的最低安装要求。还有，系统所需补丁也必须提前打上，尤其在 Linux 平台，没有成功打上补丁通常是安装失败的主要原因。如果重装数据库，在重装之前，需要干净地卸载数据库。

第 3 章　初学者最麻烦的问题之——不能干净地卸载 Oracle

为什么我们要“干净”地卸载 Oracle，只是因为重装 Oracle 时，经常遭遇失败。所谓“干净”，就是把 Oracle 从操作系统中清除得彻彻底底！



3.1　Oracle 不能完全卸载的烦恼

朋友们还记得如何安装 Oracle 吗？在安装过程中，经常会出现一些莫名其妙的问题，即使重装几次，问题依旧存在，这是比较烦人的问题，不就是一个软件安装嘛，Oracle 搞得这么麻烦。其实，好些安装失败的深层次原因是：在重装之前，我们没有干净地卸载 Oracle。若想一次性安装成功，建议大家在安装前，把 Oracle 卸载得干干净净！

本章将教会大家如何干干净净地卸载 Oracle。如果你没有安装问题，请不要浪费时间阅读本章。本章将以 Windows XP 平台上运行的 Oracle 11g 为例。



3.2　干干净净地卸载 Oracle 数据库

若想干干净净地卸载你系统中的 Oracle，请遵照本书给你提供的步骤执行。卸载 Oracle 包括下面的步骤：

- STEP 01 用 DBCA 删除数据库。
- STEP 02 删除 ASM。
- STEP 03 停止 Oracle 所有的服务（Oracle Services）。
- STEP 04 用 Oracle Universal Installer 自动删除 Oracle 的组件。
- STEP 05 手动删除 Oracle 的其余组件。STEP 05 包括如下几个小步骤：

ⓐ 删除注册表中与 Oracle 相关的内容。
ⓑ 删除与 Oracle 相关的环境变量。
ⓒ 删除与 Oracle 相关的目录或文件。
ⓓ 删除启动菜单中的 Oracle。



注意：在用 DBCA 删除数据库或者使用 Oracle Universal Installer 删除 Oracle 组件的过程中，如果出现错误，请忽略这些错误，继续执行后续的步骤。



3.2.1 用 DBCA 删除数据库

STEP 01 启动 DBCA（Database Configuration Assistant），出现如图 3-1 所示的界面。单击 下一步(N) 按钮，出现如图 3-2 所示的界面。

STEP 02 选择“删除数据库”，单击 下一步(N) 按钮，出现如图 3-3 所示的界面。

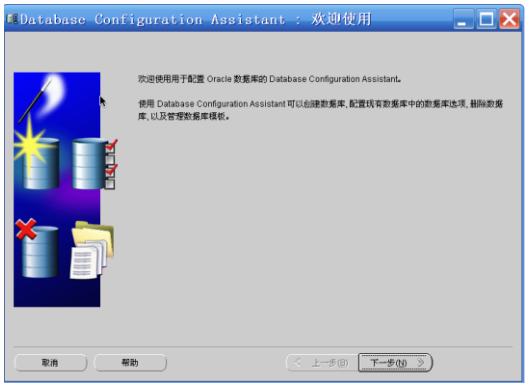


图 3-1 DBCA 欢迎界面



图 3-2 选择希望执行的操作

STEP 03 选择要被删除的数据库。

STEP 04 单击 完成(F) 按钮，出现如图 3-4 所示的对话框。

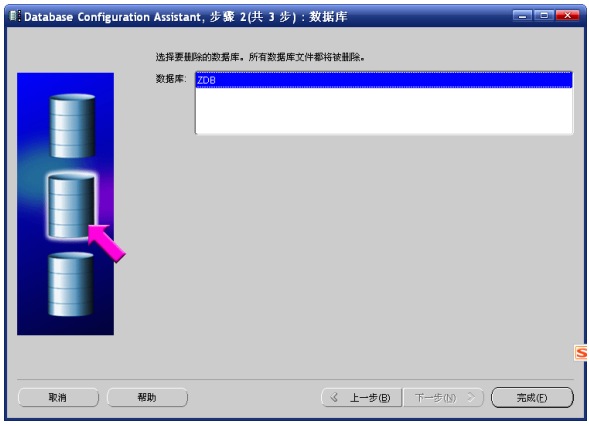


图 3-3 选择要被删除的数据库

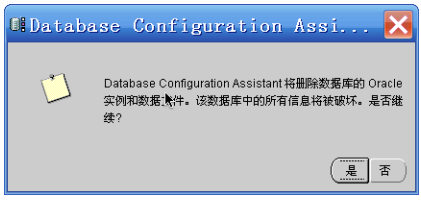


图 3-4 确认要删除数据库

STEP 05 单击 是 按钮，确定要删除数据库（图 3-4），正在删除数据库的界面如图 3-5 所示。

STEP 06 数据库删除完成的时候，弹出如图 3-6 所示的对话框。

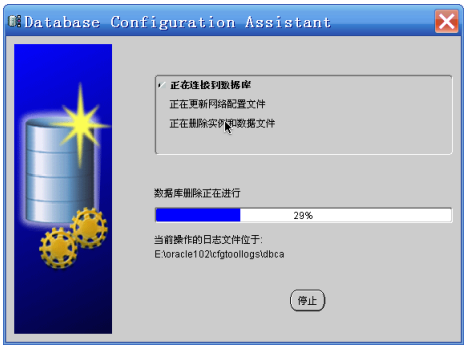


图 3-5　正在删除数据库

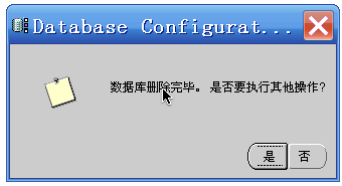


图 3-6　删除数据库完成

单击 否 按钮，不执行其他操作，数据库删除完成。如果数据库删除成功，则 Oracle 的服务 OracleService<SID>也一同被删除。

3.2.2　删除 ASM

STEP 01 设置 ASM 实例的名字。

```
C:\Documents and Settings\Administrator>set ORACLE_SID=+ASM
```

STEP 02 登录 ASM。

```
C:\Documents and Settings\Administrator>SQLPLUS /NOLOG
SQL> CONNECT SYS AS SYSASM
输入口令:
已连接。
```

STEP 03 查看哪些数据库实例正在使用 AMS 实例。

```
SQL> SELECT INSTANCE_NAME FROM V$ASM_CLIENT;
```

如果某个数据库正在使用 ASM 实例，关闭数据库或者删除表空间（该表空间是在磁盘组之上创建的）。

STEP 04 查出有哪些磁盘组。

```
SQL> SELECT NAME FROM V$ASM_DISKGROUP;
```

STEP 05 删除磁盘组。

```
SQL> DROP DISKGROUP disk_group_name INCLUDING CONTENTS;
```

STEP 06 关闭 ASM 实例。

```
SQL> SHUTDOWN
SQL> EXIT
```

STEP 07 从操作系统中删除 ASM 的服务。

```
C:\Documents and Settings\Administrator>ORADIM -DELETE -ASMSID +ASM
```

3.2.3 停止 Oracle 所有的服务（Oracle Services）

单击“开始”→“设置”→“控制面板”→“管理工具”→“服务”，停止 Oracle 所有的服务，Oracle 的服务名通常都是以“Oracle”或“Ora”开头的。

3.2.4 用 Oracle Universal Installer 自动删除 Oracle 的组件

STEP 01 启动 Oracle Universal Installer。

单击“开始”→“程序”→“Oracle 主目录（ORACLE HOME）”→“Oracle Installation Products”→“Universal Installer”，出现如图 3-7 所示的对话框。

STEP 02 单击 卸装产品(D)... 按钮，出现如图 3-8 所示的对话框。



图 3-7 OUI 欢迎界面



图 3-8 选择卸载的产品

选中你要卸载的产品。本机中存在 Oracle 的两个安装，一个是数据库服务器 OraDb11g_home3，另外一个是客户端 OraClient11g_home1。

STEP 03 选中“OraDb11g_home”，单击 删除(R)... 按钮，弹出如图 3-9 所示的对话框，确认是否要删除选中的产品。

STEP 04 单击 是 按钮，确认你要卸载这些产品，然后，开始卸载。

STEP 05 卸载完成以后，单击“关闭”→“取消”。



图 3-9 确认要被卸载的组件

3.2.5 手动删除 Oracle 的其余组件

Oracle Universal Installer 只能自动卸载 Oracle 的部分组件，其余组件需要我们手工卸载。

1. 删除注册表中与 Oracle 相关的内容

STEP 01 以管理员的身份登录操作系统（因为只有具备足够权限的用户才能修改注册表）。

STEP 02 单击桌面左下角的“开始”→“运行”，输入下面的命令打开注册表。

```
regedit
```

STEP 03 搜索注册表，找到 inst_loc，inst_loc 的值通常是 C:\Program Files\Oracle\Inventory，并记下 inst_loc 的值（在删除与 Oracle 相关的文件时将要用到这条信息）。

STEP 04 在注册表中不断搜索以 Ora、Oracle、Orcl、EnumOra 开头的值，并把它们从注册表中删除。

2．删除与 Oracle 相关的环境变量

STEP 01 打开“控制面板（Control Panel）”→“性能和维护”→“系统（System）”，选择“高级（Advanced）”→“环境变量（Environment Variables）”。

STEP 02 修改或者删除和 Oracle 相关的环境变量。

- 修改环境变量 PATH，删除其中和 Oracle 相关的路径。
- 删除环境变量 CLASSPATH。
- 删除环境变量 ORACLE_HOME、ORACLE_SID、TNS_ADMIN、JSERV、WV_GATEWAY_CFG。

3．删除启动菜单中的 Oracle

STEP 01 从启动菜单（Start Menu）中删除与 Oracle 相关的菜单并删除所有的快捷方式。

STEP 02 重启操作系统。

如果想马上重装 Oracle，这一步尤为重要。

4．删除与 Oracle 相关的目录或文件

STEP 01 删除注册表中的键 inst_loc 指向的目录。本例中是 C:\Program Files\Oracle\Inventory。

STEP 02 删除目录 C:\Program Files\Oracle。

STEP 03 删除环境变量 ORACLE_BASE 所指向的目录。

STEP 04 从\Documents and Settings\user_name\Local Settings\Temp 中删除与 Oracle 相关的目录和文件。

STEP 05 删除数据文件、控制文件与重做日志文件。总之，删除一切和 Oracle 相关的目录及文件。



本章总结

本章介绍 Oracle 的卸载。在通常情况下，参考 Oracle 的官方文档，我们可以利用 Oracle Universal Installer 卸载 Oracle，但是，那样并不能“干净”地卸载 Oracle，这是 Oracle 与其他软件不同的地方。如果遵照本章提供的步骤，就能干净地卸载 Oracle 了。

第 4 章　轻松搞定 Oracle 网络

本章将介绍和 Oracle 相关的网络问题，Oracle 网络建立在操作系统之上。配置操作系统网络是配置 Oracle 网络的第一步。在配置 Oracle 网络之前，我们需要确保操作系统的网络配置无误。

Oracle 的网络涉及两个方面，一个方面是 Oracle 的数据库服务器端；另外一个方面是 Oracle 的客户端。Oracle 网络配置的路线图如图 4-1 所示。

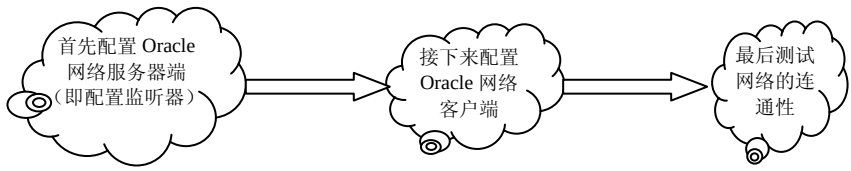


图 4-1　网络配置路线图

我们必须按照上面的步骤配置 Oracle 网络，只有在服务器端（监听器）配置成功的前提下，我们才可配置客户端。通常，配置监听器是 DBA 的工作之一。在接下来的 4.1 节，我们会介绍 Oracle 网络服务器端配置，4.2 节介绍 Oracle 客服端网络配置。

为了更好地配置 Oracle 网络，Oracle 公司提供了下面的工具，用于配置和诊断 Oracle 网络。

- 网络配置助手（Oracle Net Configuration Assistant）是一款图形化的网络管理工具，利用它可以完成 Oracle 网络的配置，包括监听器的配置、命名方法的配置、网络服务名等的配置。
- 网络管理器（Oracle Net Manager）也是一个图形化的网络管理工具，用于配置 Oracle 网络。
- 监听器控件（Listener Control Utility）是一个命令行工具，用来管理监听器，包括监听器的启动、关闭，以及查看监听器的状态等。
- 连接管理器控件（Oracle Connection Manager Control Utility）用来管理连接管理器。

4.1　轻松搞定 Oracle 网络之——数据库服务器端网络配置

本节主要介绍 Oracle 数据库服务器端网络配置（主要指监听器的配置），这是配置

Oracle 客户端的前提，因为客户端连接数据库服务器端时，好几个参数必须和服务器端指定的参数完全相同。

4.1.1 监听器的配置

配置监听器是每个 DBA 必须完成的工作之一，也是最容易出现错误的地方。只要你熟练掌握以后，配置监听器其实并不像我们大家想象的那么复杂。配置监听器有两种方法。

- **方法 1**：通过图形界面工具 Net Manager。
- **方法 2**：直接使用文本编辑器编辑监听器配置文件 listener.ora。

其实，Net Manager 最终改变的也是监听器配置文件 listener.ora 的内容。以上两种方法有着异曲同工之处。如果你熟练以后，直接编辑 listener.ora 将更加快捷！

本文将使用图形界面工具 Net Manager，以 Windows XP 为平台进行讲述。

STEP 01 启动 Net Manager。

单击“开始”→“程序”→“Oracle-Home”→“配置和移植工具（Configuration and Migration Tools）”→“Net Manager”，出现如图 4-2 所示的界面（在 Linux/UNIX 平台，使用命令 netmgr 启动 Net Manage）。



图 4-2 配置监听器首页

在配置监听器之前，我们必须理解什么是监听器。

知识点索引——监听器（LISTENER）

在 Oracle 数据库服务器中，通过一个叫“监听器”的组件接收来自客户端的连接请求。它是客户端和服务器端的中间组件。监听器是位于**服务器端**的、**独立运行的一个后台进程**，它运行在服务器端，但是独立于数据库服务器单独运行，也就是说，当数据库没有启动的时候，监听器也能独立运行。它负责对客户端传入的连接请求进行监听，并且对服务器端的连接负荷进行调整。当客户端试图建立一个到服务器端的连接时，监听器接收到客户端的请求，然后再将它交给服务器进行处理，一旦客户端和服务器建立连接，客户端和服务器以后就直接进行通信，而不再需要监听器的参与（也就是监听器将处于空闲状态）。监听器的工作原理如图 4-3 所示。

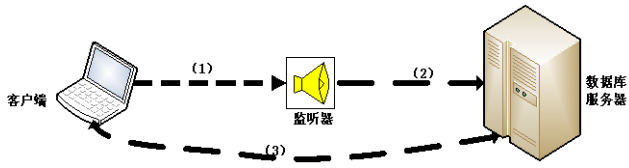


图 4-3　监听器工作原理

（1）客户端向服务器端发出连接请求，监听器监听到客户端的连接请求。

（2）把客户端的连接请求交给数据库服务器处理。

（3）客户端和服务器端建立连接。连接建立以后，服务器端和客户端直接进行通信，而不再需要监听器的参与。也就是说，监听器的职能只是负责建立客户端和服务器端的连接，它并不负责客户端和服务器端的直接通信。

例如，我们在客户端使用 SQL*Plus 登录数据库服务器以后，发出一条命令，这条命令被直接发送给数据库服务器（不通过监听器），数据库服务器处理完这条命令以后，直接把命令的执行结果返回给客户端（也不通过监听器）。

SQL*Plus 是一个客户端工具，用来和数据库进行交互，以实现对数据库的管理。有关 SQL*Plus 的更多信息，**请参考本书 5.1 节的内容。**

STEP 02　单击“本地”→“监听程序”，选择菜单“编辑”→“创建”命令，出现如图 4-4 所示的界面。在此界面中指定监听程序的名称。

在这里输入监听器的名称，可以是任何便于记忆的名字，监听器的默认名字是 LISTENER。本例中，监听器的名字保留默认值，然后单击“确定”按钮。

STEP 03　在如图 4-5 所示的界面中，单击 添加地址 按钮。



图 4-4　指定监听器名字

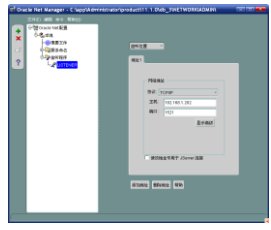


图 4-5　指定监听地址和端口号

- 指定监听器监听的协议，一般选择 TCP/IP。
- 指定监听器的主机，可以指定主机名，也可以指定 IP 地址，最好使用 IP 地址。这里的 IP 地址是数据库服务器的 IP 地址，本例中是 192.168.1.202。
- 指定端口号，Oracle 默认的端口号（Port）是 1521，我们也可以使用未被其他应用程序占用的端口号（Port），如 1522、1523 等。本例中端口号是 1521。

这里指定的 IP 地址、协议及端口号等信息将在配置客户端时被使用，客户端网络配置指定的信息必须和这些信息相同。如果两端的信息不同，Oracle 客户端将不能连接到 Oracle 服务器端，这是 Oracle 网络配置经常失败的原因！

STEP 04 单击图 4-5 正上方的下拉菜单，选择“数据库服务”，出现如图 4-6 所示的界面。

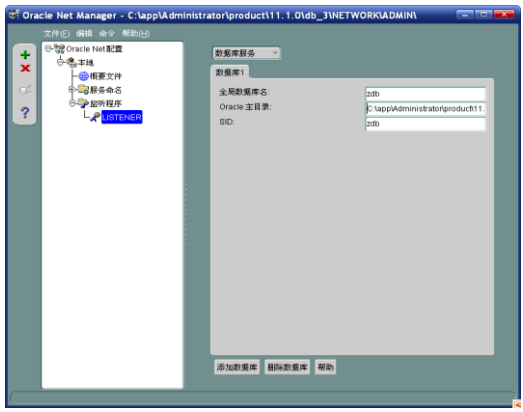


图 4-6　数据库服务设置

- 全局数据库名：输入全局数据库名，全局数据库名在单节点下和 SID 相同，本例中是 zdb。
- Oracle 主目录：输入 Oracle 的主目录（Oracle Home），Oracle 的主目录是在安装 Oracle 时由用户定义的，本例中是 C:\app\Administrator\product\11.1.0\db_3。
- SID：输入 SID，SID 就是实例名，通常是在创建数据库时定义的，本例中是 zdb。实例名和数据库名的设置在本书 2.2.3 节中有详细的介绍。

知识点索引——查找 Oracle 主目录（Oracle Home）的方法

得到 Oracle 主目录（ORACLE_HOME）的方法有以下几种：

（1）查看 Oracle 的安装日志文件，从中可以找到 Oracle 主目录（Oracle Home）的信息。安装日志的查找，**请参考本书 2.2.2 节的内容。**

（2）在注册表中进行查找。在 Windows 平台中，Oracle 主目录的位置被写进了注册表，键的名称就叫 ORACLE_HOME。通过搜索关键字 ORACLE_HOME，就可以找到 Oracle 的主目录。

（3）在 UNIX 平台中，以用户 oracle 的身份执行操作系统命令 env，在显示的信息中，环境变量 ORACLE_HOME 的值就是 Oracle 的主目录（ORACLE_HOME）。

STEP 05 保存设置。

单击“文件”→“保存网络设置”。我们在这里所做的任何修改最终都被保存到一个名叫 listener.ora 的文件中，这个文件就是“监听器配置文件”。

知识点索引——监听器配置文件 listener.ora

监听器启动的时候，会读取一个名叫 listener.ora 的文件，这个文件即为“监听器配

置文件”, 这个文件指定监听器所在的主机、监听器监听的实例、监听的端口号、监听的协议等信息。这个文件的内容类似于下面的内容：

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = zdb)
      (ORACLE_HOME = C:\app\Administrator\product\11.1.0\db_3)
      (SID_NAME = zdb)
    )
  )

LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.202)(PORT = 1521))
  )
```

LISTENER 是监听器的名字，我们也可以使用别的名字。

ORACLE_HOME 是 Oracle 的主目录。

SID_NAME 是监听器监听的数据库的实例名。本例中的实例是 zdb。

PROTOCOL 是监听器监听的协议。本例中使用的是 TCP/IP。

HOST 指定数据库服务器的主机名或者 IP 地址。本例中数据库服务器的 IP 地址是 192.168.1.202。

PORT 指定监听器使用的端口号，默认的端口号是 1521。

STEP 06 退出监听器配置。



4.1.2 监听器的启动

监听器的启动有两种方式。

■ **方式 1**：在命令行直接启动监听器。

STEP 01 登录操作系统（OS）。

如果操作系统是 Windows，用操作系统命令 cmd 打开命令行窗口；如果操作系统是 UNIX，则执行命令 su – oracle。

STEP 02 启动监听器。

```
C:\Documents and Settings\Administrator>lsnrctl start LISTENER
LSNRCTL for 32-bit Windows: Version 11.1.0.6.0 - Production on 20-7 月 -2009
16:1
2:00
Copyright (c) 1991, 2007, Oracle.  All rights reserved.
启动 tnslsnr: 请稍候...
```

```
    TNSLSNR for 32-bit Windows: Version 11.1.0.6.0 - Production
    系统参数文件为 C:\app\Administrator\product\11.1.0\db_3\network\admin\
listener.or
    a
    写入 c:\app\administrator\diag\tnslsnr\zero\listener\alert\log.xml 的日志信息
    监听: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.202)(PORT=1521)))

    正在连接到 (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.1.202)(PORT=
1521)))

    LISTENER 的 STATUS
    ------------------------
    别名                      LISTENER
    版本                      TNSLSNR for 32-bit Windows: Version 11.1.0.6.0 - Produ
    ction
    启动日期                  20-7 月 -2009 16:12:05
    正常运行时间              0 天 0 小时 0 分 4 秒
    跟踪级别                  off
    安全性                    ON: Local OS Authentication
    SNMP                      OFF
    监听程序参数文件          C:\app\Administrator\product\11.1.0\db_3\network\
admin
    \listener.ora
    监听程序日志文件          c:\app\administrator\diag\tnslsnr\zero\listener\alert\
    log.xml
    监听端点概要...
      (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.202)(PORT=1521)))
    服务摘要..
    服务 "zdb" 包含 1 个例程。
      例程 "zdb", 状态 UNKNOWN, 包含此服务的 1 个处理程序...
    命令执行成功
```

其中，LISTENER 是监听器的名字（本书 4.1.1 节指定了监听器的名字）。如果执行 lsnrctl start 命令（没有指定监听器的名称），则该命令将启动默认的监听器。默认监听器的名字是 LISTENER。

从上面的启动过程我们还可以看出有关监听器的配置信息，这些信息包括监听器所在的主机（HOST）、监听器监听的端口（Port）、支持的协议等信息。

■ **方式 2**：打开监听器管理控件后，再启动监听器。

STEP 01 登录操作系统（OS）。

如果操作系统是 Windows，用操作系统命令 cmd 打开命令行窗口；如果操作系统是 UNIX，则执行命令 su – oracle。

STEP 02 启动监听器控件。

```
    C:\Documents and Settings\Administrator>lsnrctl
    LSNRCTL for 32-bit Windows: Version 11.1.0.6.0 - Production on 20-7 月 -2009
```

```
16:1
    7:30
    Copyright (c) 1991, 2007, Oracle.  All rights reserved.
    欢迎来到 LSNRCTL，请输入"help"以获得信息。
    LSNRCTL>
```

 STEP 03 启动监听器。

```
    LSNRCTL> start LISTENER
    启动 tnslsnr：请稍候...
    TNSLSNR for 32-bit Windows: Version 11.1.0.6.0 - Production
    系统参数文件为 C:\app\Administrator\product\11.1.0\db_3\network\admin
\listene
    ……
```

其中，LISTENER 是监听器的名称。



注意：监听器命令的提示符是 LSNRCTL>。

 4.1.3 监听器的关闭

启动监听器有两种方法，关闭监听器也有两种方式。

■ **方式 1**：在操作系统命令行直接关闭。

 STEP 01 登录操作系统（OS）。

如果操作系统是 Windows，用操作系统命令 cmd 打开命令行窗口；如果操作系统是 UNIX，则执行命令 su – oracle。

 STEP 02 关闭监听器。

```
    C:\Documents and Settings\Administrator>lsnrctl stop LISTENER
```

其中，LISTENER 是监听器的名称。如果执行 lsnrctl stop 命令（没有指定监听器的名称），则该命令将关闭默认的监听器。默认监听器的名称是 LISTENER。

■ **方式 2**：登录监听器管理控件后再关闭监听器。

 STEP 01 登录操作系统（OS）。

如果操作系统是 Windows，用操作系统命令 cmd 打开命令行窗口；如果操作系统是 UNIX，则执行命令 su – oracle。

 STEP 02 登录监听器控件。

```
    C:\Documents and Settings\Administrator>lsnrctl
```

 STEP 03 关闭监听器。

```
    LSNRCTL> stop LISTENER
```

其中，LISTENER 是监听器的名称。



4.1.4 监听器的状态

有时候，我们需要知道监听器的运行状态，lsnrctl STATUS 命令可以帮助我们知道监听器的状态（包括验证监听器是否正在运行）。下面我们将给出详细的步骤。

STEP 01 登录操作系统（OS）。

如果操作系统是 Windows，用操作系统命令 cmd 打开命令行窗口；如果操作系统是 UNIX，则执行命令 su – oracle。

STEP 02 登录监听器控件。

```
C:\Documents and Settings\Administrator>lsnrctl
```

STEP 03 显示监听器的状态。

```
LSNRCTL> status LISTENER
正在连接到 (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.1.202)(PORT=
1521)))
LISTENER 的 STATUS
------------------------
别名                      LISTENER
版本                      TNSLSNR for 32-bit Windows: Version 11.1.0.6.0 -
启动日期                  20-7 月 -2009 16:23:23
正常运行时间              0 天 0 小时 10 分 54 秒
跟踪级别                  off
安全性                    ON: Local OS Authentication
SNMP                      OFF
监听程序参数文件          C:\app\Administrator\product\11.1.0\db_3\network\
admin
\listener.ora
监听程序日志文件          c:\app\administrator\diag\tnslsnr\zero\listener\
alert\
log.xml
监听端点概要...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.202)(PORT=1521)))
服务摘要..
服务 "zdb" 包含 2 个例程。
  例程 "zdb", 状态 UNKNOWN, 包含此服务的 1 个处理程序...
  例程 "zdb", 状态 READY, 包含此服务的 1 个处理程序...
服务 "zdbXDB" 包含 1 个例程。
  例程 "zdb", 状态 READY, 包含此服务的 1 个处理程序...
服务 "zdb_XPT" 包含 1 个例程。
  例程 "zdb", 状态 READY, 包含此服务的 1 个处理程序...
命令执行成功
```

命令 Status 将会列出下面有关监听器的信息：

- 监听器的名称，监听器的版本号、监听器的启动时间、运行时间、跟踪级别、监听

器配置文件 **listener**.ora 的位置等。

- 监听器的地址信息，包括主机名、端口号等。
- 监听器的实例的状态信息，实例有下面几种状态：
 - ✓ READY 表示实例可以接受连接。
 - ✓ BLOCKED 表示实例不能接受连接。
 - ✓ READY/SECONDARY 表示这是群集（Real Application Cluster）的第二个实例，准备接受连接。
 - ✓ UNKNOWN 表示监听器是静态注册的（Registered Statically）。

此外，通过监听器的 SERVICES 命令，我们还可以得到监听器更加详细的信息。如：

```
LSNRCTL> SERVICES
正在连接到 (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.1.202)(PORT=
1521))
服务摘要..
服务 "zdb" 包含 2 个例程。
  例程 "zdb", 状态 UNKNOWN, 包含此服务的 1 个处理程序...
    处理程序:
      "DEDICATED" 已建立:0 已被拒绝:0
         LOCAL SERVER
  例程 "zdb", 状态 READY, 包含此服务的 1 个处理程序...
    处理程序:
  ...
```



4.2 轻松搞定 Oracle 网络之——Oracle 客户端网络配置

本节将介绍 Oracle 客户端的网络配置。配置客户端网络时，必须确保客户端的配置信息（如 IP 地址或者主机名、实例名、协议、端口号）和服务器端的配置信息相同。



4.2.1 客户端配置

配置监听器只是配置 Oracle 网络的第一步；监听器配置完成之后，还需要配置 Oracle 客户端网络，才能实现客户端到服务器端的连接。

在配置 Oracle 的客户端之前，首先必须确认 Oracle 的客户端软件是否已经安装，这是配置的前提。除此之外，还要知道目标数据库的主机名（IP 地址）、数据库实例名（SID）、监听器的端口号等信息。

下面，我们将通过一个例子演示 Oracle 客户端的配置。远程数据库服务器（本书 2.2.3 节中创建的数据库）的信息如下。

机器名（IP）：192.168.1.202

实例名：zdb

端口号：1521

客户端的配置过程说明如下。

STEP 01 启动网络配置助手（Oracle Net Configuration Assistant）。

- 在 UNIX 平台中，在$ORACLE_HOME/bin 下运行 netca 命令。
- 在 Windows 平台中，单击“开始”→“程序”→“Oracle - HOME_NAME”→“（配置和移植工具）Configuration and Migration Tools”→“（网络配置助手）Net Configuration Assistant”，启动后的界面如图 4-7 所示。
 - ✓ 监听程序配置：用于配置监听器。
 - ✓ 命名方法配置：用于选择命名方法。
 - ✓ 本地 Net 服务名配置：用于配置本地命名方式（Local Naming）。本地命名方式将网络服务名及它们所对应的连接描述符都存储在本地的一个叫 tnsnames.ora 的配置文件中。通过该文件，能够在本地完成网络服务名的解析。
- 目录使用配置：用于设置目录命名（Directory Naming）。

选择“命名方法配置”，单击 下一步(N) > 按钮，出现如图 4-8 所示的界面。

STEP 02 选择命名方法。

“可用命名方法”中列出了可用的命名方式，有 NIS 外部命名、DCE CDS 外部命名、轻松连接命名、本地命名、目录命名。在生产环境中，用得最多的是“本地命名”。



图 4-7　网络配置助手首页



图 4-8　命名方法选择

在“可用命名方法”下，选中“本地命名”，然后单击 > 按钮，则本地命名方法将会被添加到“选定的命名方法”列表中。也就是说，我们将使用“本地命名”方法。单击 < 按钮可以去除已选择的“命名方法”。

当然，我们也可以混合使用多种命名方法，如同时使用“本地命名”和“目录命名”两种方法。当客户端连接服务器时，如果同时使用多种命名方法，Oracle 会按照命名方法出现的顺序进行一一尝试，直到能正确连接数据库服务器为止。 和 按钮用于调整命名方法的顺序。

STEP 03 命名方法设置完成。

单击 下一步(N) > 按钮，出现如图 4-9 所示的界面。

单击 下一步(N) > 按钮，命名方法配置完成，我们又回到如图 4-7 所示的界面。

STEP 04 配置本地命名。

在 如图 4-7 所示的界面中选择“本地 Net 服务名配置”，然后单击 下一步(N) > 按钮，出现如图 4-10 所示的界面。



图 4-9　命名方法配置完成



图 4-10　网络服务名配置选择

在这里，我们可以选择“添加”、“删除”或“重新配置”网络服务名。选择“添加”，然后单击 下一步(N) 按钮，出现如图 4-11 所示的界面。

STEP 05 设置服务名（SID）。

这里的“服务名”其实就是实例名（SID）。本例中，在“服务名”处输入 zdb（以前创建的实例），然后单击 下一步(N) 按钮，出现如图 4-12 所示的界面。

STEP 06 选择协议。



图 4-11　指定网络服务名



图 4-12　选择协议

Oracle 支持的协议有：

- TCP/IP
- TCP/IP with SSL（TCPS）
- Named Pipes（NMP）
- SDP
- IPC

Oracle 的网络是基于操作系统的网络配置，在配置 Oracle 网络之前，我们必须安装和配置好操作系统（OS）的网络。TCP/IP 是应用最广泛的通信协议，通常我们选择 TCP/IP。选择“TCP”，然后单击 下一步(N) 按钮，出现如图 4-13 所示的界面。

STEP 07 设置主机名和端口号。

- 主机名：这里的“主机名”是数据库服务器所在的**机器名**，也可以是数据库服务器的 **IP 地址**。本例中的主机是 192.168.1.202。
- 端口号：这里的端口号是监听器使用的端口号，1521 是默认的端口号。本例中的

端口号是 1521。

这里的主机名、选择的协议、端口号都必须和服务器端的**相同**，否则客户端不能成功地连接到数据库服务器。如果你不清楚这些信息，请询问你的 DBA。

单击 下一步(N) 按钮，出现如图 4-14 所示的界面。

STEP 08 测试选择。



图 4-13　指定主机名和端口号



图 4-14　测试选择

这里选择是否要对上面的网络配置进行测试，看看配置是否正确，以及是否能够成功连接到数据库服务器。

选择“是，进行测试”，然后单击 下一步(N) 按钮，通常会出现如图 4-16 所示的界面。

STEP 09 测试结果。

如果测试成功，将会显示如图 4-15 所示的信息；如果测试失败，Oracle 会显示失败的原因（如图 4-16 所示）。测试时，如果提示用户名和密码错误，我们可以重新设置用户名和密码，单击“更改登录”按钮，出现如图 4-17 所示的界面。



图 4-15　测试成功



图 4-16　测试失败

STEP 10 设置用户名和密码。

这里的用户是数据库用户，不是操作系统的用户。输入用户名和密码（本例中用户名是 system，密码是 zero），单击“确定”按钮。如果测试成功，将会出现如图 4-15 所示的界面。

STEP 11 设置网络服务名。

在如图 4-15 所示的界面中，单击 下一步(N) 按钮，出现如图 4-18 所示的界面。



图 4-17　设置用户名和密码



图 4-18　指定网络服务名

在这里输入**网络服务名**，以后我们将使用**网络服务名**来连接远程数据库。网络服务名

的名称可以任意指定，但是，从应用的角度考虑，我们应该指定一个便于记忆的名称。



知识点索引——网络服务名

当我们访问远程数据的时候，都需要输入类似于下面的**连接描述符**，这些信息描述了我们要连接的数据库。

```
(DESCRIPTION =
   (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.202)(PORT = 1521))
   )
   (CONNECT_DATA =
     (SERVICE_NAME = zdb)
   )
 )
```

这些信息非常冗长，如果用户每次访问远程数据库时都要输入那么冗长的信息，那将是生命的极度浪费。为此，DBA 可以给这些冗长的字符串（连接描述符）起一个“别名”。在 Oracle 中，我们把这样的别名叫“**网络服务名（Net Service Name）**”。如果配置好了网络服务名，当我们每次连接远程数据库时，无须输入那么冗长的信息，只需输入网络服务名，便可轻轻松松登录远程的数据库服务器。

网络服务名存储在名为 tnsnames.ora 的文件中，可以把该文件复制到网络中的所有客户机和服务器上。

在服务器上，tnsnames.ora 文件的位置由环境变量 TNS_ADMIN 指定。tnsnames.ora 通常被存放在下面的路径：

- ✓ $ORACLE_HOME/network/admin（UNIX 平台）
- ✓ ORACLE_HOME\network\admin（Windows 平台）

其中，ORACLE_HOME 表示 Oracle 的主目录（Oracle Home）。

下面我们将举例说明使用网络服务名登录数据库给我们带来的益处。在本例中，将要连接的目标数据库的主机 IP 是 192.168.1.202，端口号是 1521，实例名是 zdb。



示例 1　使用“连接字符串”登录远程数据库

```
C:\Documents and Settings\Administrator>sqlplus system/zero@192.168.
1 .202:1521/zdb

SQL*Plus: Release 11.1.0.6.0 - Production on 星期二 7 月 21 14:10:24 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
连接到:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
```

- ✓ system 是数据库用户的名字。

✓ zero 是用户 system 的密码。

✓ 192.168.1.202 是数据库服务器的 IP 地址。

✓ 1521 表示端口号。

✓ zdb 表示实例名。

从上面可以看出，每次登录远程数据库服务器时，我们都需要输入远程数据库服务器的主机名、端口号和实例名等信息。这项工作很烦琐，如果在实际应用中，数据库服务器的数量很多，对于普通用户来说，记住每台数据库服务器的主机名（IP）、端口号等信息也不是一件容易的事，这就凸显了使用**连接字符串**登录数据库的弊端。



示例 2　使用“网络服务名”登录远程数据库

在本例中，客户端网络配置文件 tnsnames.ora 的内容如下，这个文件定义了网络服务名 MYZDB。

```
MYZDB =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.202)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = zdb)
    )
  )
```

✓ PROTOCOL 用于指定使用的协议。本例中使用的是 TCP/IP。

✓ HOST 指定数据库服务器的主机名或者 IP 地址。本例中数据库服务器的 IP 地址是 192.168.1.202。

✓ PORT 指定监听器使用的端口号，默认的端口号是 1521。

✓ SERVICE_NAME 指定请求数据库的实例名。本例中请求的实例是 zdb。

使用网络服务名 myzdb 登录远程数据库。

```
C:\Documents and Settings\Administrator>sqlplus system/zero@myzdb
SQL*Plus: Release 11.1.0.6.0 - Production on 星期二 7月 21 14:20:51 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
连接到:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
```

从上面的过程可以看出，使用“网络服务名”登录数据库的时候，我们无须知道远程数据库服务器的主机名（IP 地址）、端口号、实例名等信息，我们只需输入服务名 myzdb 便能轻轻松松登录数据库。

对比上面两个例子可以看出，使用**连接字符串**登录数据库比较麻烦，且需要记忆

的东西比较多，特别是对不熟悉 Oracle 的用户；使用**网络服务名**登录数据库比较方便，只需输入**网络服务名**便轻松搞定！在 Oracle 的实际应用中，基本上都是使用**网络服务名**来连接数据库的。

输入网络服务名 myzdb，单击 下一步(N) 按钮，出现如图 4-19 所示的界面。

STEP 12 选择是否要配置另外一个网络服务名。

到此，我们已经完成网络服务名的配置。如果还想配置另外一个网络服务名，请选择“是”；如果不想，请选择“否”。

单击 下一步(N) 按钮，出现如图 4-20 所示的界面。



图 4-19　是否配置其他网络服务名



图 4-20　网络服务名配置完成

STEP 13 配置完毕。

单击 下一步(N) 按钮，我们又回到了如图 4-7 所示的界面，在这里，我们还可以进行其他网络配置。

当然，我们也可以使用图形界面工具 Net Manager 来配置 Oracle 的客户端。其原理和使用网络配置助手（Net Configuration Assistant）来配置 Oracle 的网络客户端的原理差不多。



知识点索引——手工编辑文件 tnsnames.ora

我们可以通过直接编辑文件 tnsnames.ora 来实现删除、添加、修改网络服务名的操作。编辑文件 tnsnames.ora 和使用网络配置助手（Net Configuration Assistant）实现的效果相同。**使用网络配置助手（Net Configuration Assistant）的实质就是修改文件 tnsnames.ora。**



找到文件 tnsnames.ora。

文件 tnsnames.ora 的位置，**请参考本节“知识点索引——网络服务名”的内容。**



用文本编辑器打开文件 tnsnames.ora。

在本例中，文件的内容如下所示：

```
MYZDB =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.202)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = zdb)
    )
  )
```



创建新的网络服务名 KK。

复制上面的内容，在该文件的后面进行粘贴。只对复制的内容进行修改，把 MYZDB 替换成 KK，修改主机名（192.168.1.12）、端口号（1522）、实例名（nbo）。新的 tnsnames.ora 文件内容如下所示：

```
MYZDB =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.202)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = zdb)
    )
  )

KK =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.12)(PORT = 1522))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = nbo)
    )
  )
```

4.2.2 测试客户端到服务器的连接

经过长途跋涉（Oracle 软件的安装、数据库创建、监听器配置、客户端配置），相信大家已经有些疲倦，坚持一下，激动人心的一刻即将来临。

为了测试客户端到服务器的网络是否通畅，我们使用 Oracle 提供的 tnsping 命令。tnsping 是一个在操作系统命令行执行的命令。接下来，我们将测试在此之前的网络配置是否正确，同时，也想通过这个例子展示 tnsping 的用法。

STEP 01 登录操作系统（OS）。

如果操作系统是 Windows，用操作系统命令 cmd 打开命令行窗口；如果操作系统是 UNIX，则执行命令 su – oracle。

STEP 02 执行 tnsping 命令。

```
C:\Documents and Settings\Administrator>tnsping myzdb
TNS Ping Utility for 32-bit Windows: Version 11.1.0.6.0 - Production on 21-7
月 -2009 15:26:53
Copyright (c) 1997, 2007, Oracle.  All rights reserved.
已使用的参数文件:
C:\ora_client\product\11.1.0\client_2\network\admin\sqlnet.ora
已使用 TNSNAMES 适配器来解析别名
Attempting to contact (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL
= TCP)
(HOST = 192.168.1.202)(PORT = 1521))) (CONNECT_DATA = (SERVICE_NAME = zdb)))
OK (60 毫秒)
```

在本例中，myzdb 是我们之前配好的一个**网络服务名**。这个网络服务名指向远程的数据库。如上面所示，如果客户端能够成功连接到服务器，tnsping 命令的执行结果将会显示“OK”，并报告这个过程消耗了多少时间（本例中消耗 60 毫秒）；如果不能成功连接到远程的服务器，tnsping 命令将会返回详细的错误信息。

4.2.3 登录远程数据库

人类进入 21 世纪，信息技术让网络延伸到地球的每一个角落，无论何时（Anytime），无论何地（Anywhere），我们都可以触摸到地球的另外一个角落。无论在中国还是在美国，我们都可以访问到远在天涯海角的数据库，就好像它近在咫尺。

要对远程数据库进行访问，必须先登录远程数据库，我们可以使用 SQL*Plus，也可以使用企业管理器（Oracle Enterprise Manager，OEM）登录远程数据库。下面，我们将演示使用 SQL*Plus 如何登录远程数据库。本例中，192.168.1.202 是一台远程数据库。

STEP 01 登录操作系统（OS）。

如果操作系统是 Windows，用操作系统命令 cmd 打开命令行窗口；如果操作系统是 UNIX，则执行命令 su – oracle。

STEP 02 打开 SQL*Plus，登录数据库。

```
C:\Documents and Settings\Administrator>sqlplus
SQL*Plus: Release 11.1.0.6.0 - Production on 星期二 7月 21 15:46:47 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
请输入用户名:  system@myzdb
输入口令:
连接到:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
```

哈哈……远程数据库登录成功，功夫不负有心人啊。

其中 sqlplus 是命令，用于打开 SQL*Plus。在 system@ myzdb 中，system 是数据库的用户名；@表示要登录哪个数据库；myzdb 是我们前面配置好的**网络服务名**，它指向目标数据库（192.168.1.202）。



4.3 轻松搞定 Oracle 网络之——Oracle 网络故障攻略

引起客户端不能访问数据库的原因很多，有软件问题，也有硬件问题；有操作系统本身的问题，也有应用程序的问题，也可能是防火墙的问题。但是，在多数情况下都是用户配置不当所致。无论什么网络错误，我们都可以按照下面的顺序和原则去侦测网络问题。

STEP 01 ping 目标主机。

用操作系统命令 ping 目标主机，看看目标主机是否存在。

```
C:\Documents and Settings\Administrator>ping 192.168.200.38
Pinging 192.168.200.38 with 32 bytes of data:
Request timed out.
Ping statistics for 192.168.200.38:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss)
```

一般情况下，我们基本上可以从这里断定问题是否与操作系统有关。

如果 ping 的结果是“**Request timed out**”，表示客户端到目标主机（192.168.200.38）的网络不通。这时候，需要解决**操作系统**这一层的网络连接问题。如果不是操作系统的网络问题，接下来，我们可以执行 tnsping 命令查看问题是否出在 Oracle 配置这一层。

ping 不通可能有下面的原因：

- 客户端的网卡物理故障
- 客户端的网卡设置不当
- 客户端的防火墙限制
- 网线断裂

- 服务器端的网卡物理故障
- 服务器端的网卡设置不当
- 服务器端的防火墙限制
- ……

STEP 02 执行 tnsping 命令。

```
tnsping 网络服务名
```

tnsping 命令主要用于测试**客户端**是否能够连接到**监听器**。

有关 tnsping 命令的使用，**请参考本书 4.2.2 节的内容**。

如果 tnsping 命令报告错误，我们需要执行 STEP 03。

STEP 03 查看监听器状态。

■ 登录监听器控件。

```
C:\Documents and Settings\Administrator>lsnrctl
```

■ 显示监听器的状态。

```
LSNRCTL> status
```

从这里，我们可以知道监听器的运行状态。如果监听器运行不正常，或者客户端设置的参数与监听器的参数不匹配，都会导致网络故障。

有关监听器状态的详细信息，**请参考本书 4.1.4 节的内容**。

STEP 04 查看监听器配置文件（listener.ora）。

监听器启动的时候，会读取一个名叫 listener.ora 的文件，这个文件即为“监听器配置文件”，它指定监听器所在的主机、监听器监听的实例名、监听的端口号、监听的协议等信息。这个文件的内容类似于下面的内容：

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = zdb)
      (ORACLE_HOME = C:\app\Administrator\product\11.1.0\db_3)
      (SID_NAME = zdb)
    )
  )

LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.202)(PORT = 1521))
  )
```

- LISTENER 是监听器的名字，我们也可以使用别的名字。
- ORACLE_HOME 是 Oracle 的主目录。
- SID_NAME 是监听器监听的数据库的实例名。
- PROTOCOL 是监听器监听的协议。

- HOST 指定数据库服务器的主机名或者 IP 地址。
- PORT 指定监听器使用的端口号，默认的端口号是 1521。

需要仔细比对客户端的参数是否和这些参数匹配。

在确定服务器端配置无误后，我们需要查看客户端的配置。

STEP 05 查看客户端网络配置文件 tnsnames.ora。

客户端网络配置文件 tnsnames.ora 的样例如下所示（这个文件定义了网络服务名 MYZDB）：

```
MYZDB =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.202)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = zdb)
    )
  )
```

- PROTOCOL 用于指定使用的协议。
- HOST 指定数据库服务器的主机名或者 IP 地址。
- PORT 指定监听器使用的端口号，默认的端口号是 1521。
- SERVICE_NAME 指定请求的数据库的实例名。

需要仔细比对这些参数与监听器的参数是否匹配。

STEP 06 查看 Oracle 网络的日志文件。

如果完成上述步骤，仍然不能诊断出网络故障，我们可以查看 Oracle 网络的日志文件，从这些文件中，我们能够获得更多的信息，这些文件如下：

- listener.log　　Listener（监听器日志文件）
- sqlnet.log　　Client or Database Server（存在于客户端和服务器端）
- instance-name_pid.log　Oracle Connection Manager Listener（有关连接管理器的日志文件）
- ……



本章总结

本章完整介绍了 Oracle 的网络配置（服务器端和客户端）。要掌握 Oracle 的网络配置，只需掌握两个要点：一是要学会配置监听器，并要懂得监听器的基本原理；二是在配置客户端时，必须保证客户端的配置信息和服务器端的配置信息匹配。另外，还需要注意防火墙对 Oracle 网络的影响。

第 5 章　访问远程数据库之——工具介绍

数据库一旦创建后，我们可以远程对数据库进行访问，就好像数据库服务器就摆放在我们面前。用于访问数据库的工具有很多，有 Oracle 公司自己提供的工具（如 SQL*Plus 和 OEM），还有第三方的工具（如 DbVisualizer、TOAD），第三方的工具有其独特的功能，这些功能更强调实际应用，它们可以更高效地辅助我们完成数据库的管理任务。

5.1　字符界面管理工具之——SQL*Plus

SQL*Plus 是 DBA 的最爱！它是一个字符界面的工具，功能强大，几乎可以完成大部分的数据库管理任务。在 SQL*Plus 中，你可以尽情展示数据库管理的技巧。

5.1.1　启动 SQL*Plus

SQL*Plus 的启动过程说明如下。

STEP 01　打开操作系统的命令行。

- Windows 平台：单击“开始”→“运行”，输入 cmd，进入操作系统的命令行。
- UNIX 平台：打开一个终端，以用户 oracle 的身份登录操作系统。

STEP 02　在命令行中直接输入 sqlplus。

- c:\> sqlplus　　（Windows 平台）
- $ sqlplus　　（UNIX 平台）

STEP 03　输入用户名和密码，登录数据库，出现如图 5-1 所示的界面。

```
SQL*Plus: Release 11.1.0.6.0 - Production on 星期三 7 月 22 21:31:25 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
请输入用户名:  system@myzdb
输入口令:
```

system 是数据库的用户名，zero 是 system 的密码，myzdb 是我们在第 4 章中创建的网络服务名。

至此，我们已经进入了 SQL*Plus。在此窗口中，可以输入命令，执行数据库的管理任务。

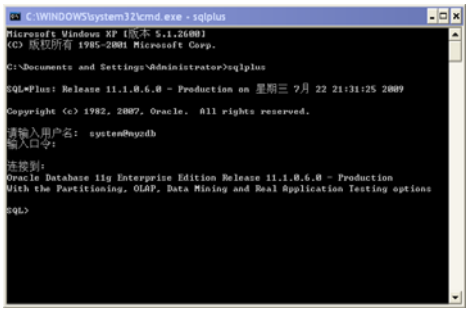


图 5-1　SQL*Plus 的首页



知识点索引——什么是 SQL*Plus

SQL*Plus 是 Oracle 公司推出的一个交互式管理工具，它用于数据库的管理。数据库的所有管理任务，通过 SQL*Plus 几乎都可以实现。Oracle 公司还有其他图形界面管理工具——Oracle 企业管理器（Oracle Enterprise Manager），它也可用于数据库的管理。与企业管理器相比，SQL*Plus 有其独特的地方，企业管理器能够实现的功能，SQL*Plus 亦都能实现；但 SQL*Plus 能实现的功能，企业管理器未必能实现。SQL*Plus 通过命令行对数据库进行管理，这增加了数据库管理的灵活性。如果想成为一个优秀的 DBA，Oracle 推荐你使用 SQL*Plus。

SQL*Plus 有自己的**命令**和**环境**。特别注意的是，这里的**命令**是 SQL*Plus 所特有的，**这种命令只能在 SQL*Plus 中执行**，我们把这种命令叫 **SQL*Plus 专有命令（或者叫"SQL*Plus 命令"）**。SQL*Plus 专有命令（SQL*Plus Commands）与 SQL 语句不同，SQL 语句可以在许多工具中执行，而 **SQL*Plus 命令**只能在 SQL*Plus 中执行。

SQL*Plus 有如下功能：

- ✓ 对查询结构进行格式化、存储、打印。
- ✓ 检查表和对象的定义。
- ✓ 开发和执行批处理脚本。
- ✓ 执行数据库的管理。



注意：SQL*Plus 的命令行提示符是 SQL>。



5.1.2　在 SQL*Plus 中执行代码

可以在 SQL*Plus 中执行的命令有：SQL 语句、PL/SQL 块、SQL*Plus 命令（SQL*Plus Commands）和操作系统命令。

■ 执行 SQL 语句

在提示符（SQL>）下输入 SQL 语句，SQL 语句结束有三种方式：以分号结束（;）；在单独的一行，输入斜杠（/）；以一行空白行结束。输入分号（;）或者斜杠（/），并按回车键将执行输入的 SQL 语句。如在 SQL*Plus 中执行下列 SQL 语句：

```
SQL> SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY
  2  FROM EMP_DETAILS_VIEW WHERE SALARY > 12000;
```

SQL 语句的执行结果是：

```
EMPLOYEE_ID LAST_NAME                JOB_ID          SALARY
----------- ------------------------ ---------- ----------
            201 Hartstein       MK_MAN          13000
            100 King                AD_PRES         24000
2 rows selected.
SQL>
```

本例查询工资高于 12000 元的员工。

■ 执行 PL/SQL 块

PL/SQL 块（Block）是以 DECLARE 或 BEGIN 开头并以 END 结束的一段程序代码。可以直接在 SQL*Plus 中执行 PL/SQL 块。

以下的代码段往表 A4 中插入 1000 条记录。

```
SQL> BEGIN
  2    FOR v_Counter IN 1..1000 LOOP
  3      INSERT INTO A4
  4        VALUES ('21212fdfdsfdsf');
  5    END LOOP;
  6   COMMIT;
  7  END;
  8  /
```

END 之后的分号（;）表示 PL/SQL 块的结束，它并不会让 Oracle 执行 PL/SQL 块。在新的行中输入斜杠（/）将执行 PL/SQL 块。

■ 执行 SQL*Plus 命令

在输入 SQL*Plus 命令后，直接按回车键将执行输入的 SQL*Plus 命令。

STEP 01 执行 SQL*Plus 命令 COLUMN，对列 **SALARY** 的输出格式化。

```
SQL> COLUMN SALARY FORMAT $99,999 HEADING 'MONTHLY SALARY'
```

STEP 02 再次执行执行过的 SQL 语句。

```
SQL> SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY
  2  FROM EMP_DETAILS_VIEW WHERE SALARY > 12000;
```

SQL 语句的执行结果是：

```
EMPLOYEE_ID LAST_NAME                JOB_ID     MONTHLY SALARY
```

```
---------- ------------------------ ---------- -------------
       201 Hartstein                    MK_MAN          $13,000
       100 King                       AD_PRES         $24,000
2 rows selected.
```

从结果可以看出，同一条 SQL 语句，输出的格式却不相同，这是 SQL*Plus 命令 COLUMN 导致的结果。

■ 执行操作系统命令

有时，需要在 SQL*Plus 中执行操作系统命令，SQL*Plus 提供了这样的功能。具体的格式如下：HOST OS COMMANDS，其中 HOST 是关键字，后面跟着操作系统命令，直接按回车键将执行输入的命令。

```
SQL> host dir d:\
```

执行结果是：

```
abc.sql  afiedt.buf  createview_UNIX.sql  ifile  init.ora  log  tmp
```

在本例中，在 SQL*Plus 中执行操作系统命令 dir，列出目录 d:\下所有的文件。

5.1.3 关闭 SQL*Plus

如果不再使用 SQL*Plus，请退出 SQL*Plus 以节省系统资源。退出 SQL*Plus 有下面几种方式：

- 在 SQL*Plus 提示符下，输入 exit 或者 quit。
- 按 Ctrl+D（UNIX 平台）组合键或按 Ctrl+Z（在 Windows 平台）组合键。

5.1.4 最有用的 SQL*Plus 命令（SQL*Plus Commands）

SQL*Plus 的专有命令很多，本书只节选部分命令进行讲解，如果想知道所有专有命令的详细用法，请参考 Oracle 的官方文档——《SQL*Plus User Guide and Reference》。

1．登录，注销命令

■ 命令：CONNECT（连接数据库）

语法：CONNECT username/password@网络服务名 SYSOPER | SYSDBA

用法：登录数据库。

“username/password”指定用于登录数据库的用户名和密码。

“@网络服务名”指定登录到哪个数据库。

“SYSOPER | SYSDBA”表示以超级用户的身份登录数据库。

例子：`SQL> CONNECT system/wy@myzdb`

■ 命令：DISCONNECT（退出登录）

语法：DISC[ONNECT]

用法：退出数据库登录，但并不退出 SQL*Plus。

例子：

```
SQL> DISCONNECT
```

2．编辑命令

■ 命令：APPEND（追加）

语法：A[PPEND] text

用法：把文本添加到当前行的末尾。

例子：`SQL>APPEND CITY`

■ 命令：CHANGE（修改）

语法：C[HANGE] /旧字符串/新字符串

用法：将当前行中的“旧字符串”替换成“新字符串”。

例子：`SQL>CHANGE /CLERK/SH_CLERK`

■ 命令：CLEAR（清除）

语法：CL[EAR] 选项

选项：

BRE[AKS]	（清理分组定义）
BUFF[ER]	（清理 SQL*Plus 缓冲区中的内容）
COL[UMNS]	（清除列的显示属性，把它设成默认值）
COMP[UTES]	（清理计算设置）
SCR[EEN]	（清理屏幕）
SQL	（清理 SQL 缓冲区中的内容）
TIMI[NG]	（清除定时）

用法：清除某项设置或者内容。

例子：`SQL>CLEAR COLUMNS`

■ 命令：DEL（删除）

语法：DEL [n | n m | n * | n LAST | * | * n | * LAST | LAST]

n	删除行 n
n m	删除从行 n 到行 m 之间的行
n *	删除从行 n 到当前行之间的所有行
n LAST	删除从行 n 到最后一行之间的所有行
*	删除当前行
* n	删除从当前行到行 n 之间的所有行
* LAST	删除从当前行到最后一行之间的所有行
LAST	删除最后一行

用法：删除缓冲区中指定的行。

例子：`SQL>DEL（删除缓冲区中的当前行）`



知识点索引——SQL 缓冲区（SQL Buffer）

SQL 缓冲区用于存储用户输入的 SQL 语句或者 PL/SQL 块。特别注意的是 SQL 缓冲区不存放 SQL*Plus 专有命令。

SQL 缓冲区中的内容一直保持，直到被下一条 SQL 语句或者 PL/SQL 所覆盖。我们可以通过 LIST 命令查看 SQL 缓冲区中的内容，也可以编辑缓冲区中的内容。

■ 命令：INPUT（添加行）

语法：I[NPUT] [text]

用法：在当缓冲区中的当前行再添加一行。

例子：`SQL>INPUT sfsf`

■ 命令：LIST（显示行的内容）

语法：L[IST] [n | n m | n * | n LAST | * | * n | * LAST | LAST]

n	显示行 n 的内容
n m	显示从行 n 到行 m 之间的行的内容
n *	显示从行 n 到当前行之间的所有行的内容
n LAST	显示从行 n 到最后一行之间的所有行的内容
*	显示当前行的内容
* n	显示从当前行到行 n 之间的所有行的内容
* LAST	显示从当前行到最后一行之间的所有行的内容
LAST	显示最后一行的内容

用法：显示指定行的内容。

例子：显示第二行的内容，命令如下所示。

```
SQL> LIST 2
2* iiii
```

3．其他有用的 SQL*Plus 命令

■ 命令：PROMPT

语法：PRO[MPT] [text]

用法：显示文本信息。

例子：`SQL> PROMPT Please enter a valid department`

```
Please enter a valid department
```

■ 命令：DESCRIBE

语法：DESC[RIBE] {[schema.]object[@db_link]}

用法：列出表、视图、同义词的列；列出函数及存储过程的定义。

例子：`SQL> DESC A`

```
名称                                      是否为空？类型
----------------------------------------- -------- ------------
A                                         NUMBER(38)
```

这个例子列出表 A 的结构。

■ 命令：DEFINE

语法：DEF[INE] [variable] | [variable = text]

用法：定义一个变量，并把一个字符值赋给它；或者显示变量的值。

例子：定义一个变量。

```
SQL> DEFINE myid=22
```

显示变量的值和变量的类型。

```
SQL> DEFINE myid
DEFINE MYID = "22" (CHAR)
```

■ 命令：ACCEPT

语法：ACC[EPT] variable [NUM[BER] | CHAR | DATE | BINARY_FLOAT | BINARY_DOUBLE] [FOR[MAT]format] [DEF[AULT] default] [PROMPT text|NOPR[OMPT]] [HIDE]

用法：提示用户输入值，并把这个值存放在替换变量中。

例子：`SQL> ACCEPT pswd CHAR PROMPT '用户名: '`

其中 system 是用户名；pswd 是变量名；CHAR 是变量的数据类型。

■ 命令：SET

语法：SET system_variable value

其中 system_variable 表示环境变量的名称；value 表示环境变量的值。

用法：设置 SQL*Plus 的环境。

例子：`SQL> SET TIMING ON;`



知识点索引——SQL*Plus 环境变量

SQL*Plus 环境变量，也叫 SQL*Plus 系统变量（System Variables），它控制 SQL*Plus 的行为，如列的宽度、是否允许 SQL 语句的自动提交、是否显示记录编号等。

可以通过 SET 命令设置 SQL*Plus 环境变量，也可以通过 SHOW 命令显示环境变量的当前值。

■ 命令：RUN（运行）

语法：R[UN]

用法：运行 SQL 缓冲区中的 SQL 语句。

例子：`SQL> run`



5.1.5 SQL*Plus 使用技巧

本节将展示 SQL*Plus 的一些使用技巧。

1. 知道一条 SQL 语句的执行时间

如何能知道一条 SQL 语句的执行速度呢？SQL*Plus 的系统变量 TIMING 可以用来跟踪每条 SQL 语句的执行时间。当一条 SQL 语句执行完成，在返回结果集后会自动显示 SQL 语句的执行时间。得到 SQL 语句执行时间的过程如下：



启动定时跟踪。

```
SQL> SET TIMING ON;
```

SQL*Plus 的 SET 命令将 TIMING 环境变量设成 ON，启动对每个 SQL 语句的跟踪。也就是说，从这里开始，每条执行的 SQL 语句都会自动返回执行时间。

STEP 02 执行 SQL 语句，查看 SQL 语句耗费的时间。

```
SQL> SELECT * FROM student;
```

SQL 语句的执行结果是：

```
NAME                                     TELEPHONE
---------------------------------------- --------------------
小王                                     13913307891
小张                                     13913307891
小李                                     13913307891
小不点                                   13913307891
mary                                     13913307891
已用时间:  00: 00: 00.05
```

这条 SQL 语句耗费 5 毫秒。

STEP 03 关闭定时跟踪。

当不再需要跟踪 SQL 语句的执行时间时，关闭跟踪功能。SQL*Plus 的 SET 命令将 TIMING 环境变量设成 OFF，关闭对每个 SQL 语句的跟踪。也就是说，从这里开始，每条执行的 SQL 语句都不会返回执行时间。

```
SQL> SET TIMING OFF;
```

2．通过 SQL*Plus 导入写好的 SQL 脚本

有时候，我们需要运行一系列任务，我们可以把与这些任务相关的 SQL 语句放在一个脚本文件中，然后利用 SQL*Plus 提供的命令，**批量运行** SQL 脚本文件（也就是命令的批量执行）。脚本可以包含一些格式化命令（如 SPOOL、SET TIMING ON 等）。

SQL*Plus 的 start 命令或者@命令都可以用来运行一个脚本文件（start 文件名，@文件名）。在一个脚本文件中可以用 start 命令调用另外一个脚本（如 start　sqles，其中 sqles 也是一个脚本文件），这样可以实现多个脚本的嵌套调用。

以下是用@运行一个脚本文件的例子。

STEP 01 以下是脚本 ch05_1.SQL 的内容。

```
SELECT sysdate FROM dual;
drop table student;
CREATE TABLE student
(name VARCHAR2(50),telephone VARCHAR2(20));
insert into student
values('小王','13913307891');
insert into student
values('小张','13913307891');
insert into student
values('小李','13913307891');
insert into student
```

```
values('小不点','13913307891');
insert into student
values('mary','13913307891');
COMMIT;
```

- 第 1 条语句显示当前的系统日期。
- 第 2 条语句删除学生信息表，删除的目的是为了重建。
- 第 3 条语句是重建学生信息表（student）。
- 第 4 条语句是把学生“小王”的信息插入学生信息表（student）中。
- ……

STEP 02 用@命令运行文件 ch05_1.SQL。

```
SQL>@F:\script\ch05_1.SQL
```

执行结果是：

```
SYSDATE
------------
23-JUL-09
1 row selected.
Table created.
1 row created.
1 row created.
  ……
Commit complete.
SQL>
```



温馨提示：要运行的脚本文件要和 SQL*Plus 在同一台机器上。运行脚本时，一定要注意脚本文件名的后缀，如果没有指定脚本文件的后缀，SQL*Plus 可能会报告找不到该文件。

3．DBMS_OUTPUT.PUT_LINE 不能输出结果的问题

SQL*Plus 的环境变量 SERVEROUTPUT 影响 DBMS_OUTPUT.PUT_LINE 的输出，执行 DBMS_OUTPUT.PUT_LINE 之前，要打开 SERVEROUTPUT。

STEP 01 执行下面的块，输出内容'HELLO ZERO'。

```
SQL> DECLARE
  2       k VARCHAR2(100);
  3   BEGIN
  4       k:='HELLO ZERO';
  5       DBMS_OUTPUT.PUT_LINE(k);
  6   END;
  7  /
PL/SQL procedure successfully completed.
```

没有任何输出。

STEP 02 打开文本输出。

```
SQL> SET SERVEROUTPUT ON
```

STEP 03 用“/”重新执行上面的 PL/SQL 块，打印出结果。

```
SQL> /
HELLO ZERO
PL/SQL procedure successfully completed.
```

4．终止正在运行的 SQL 语句

如果一条 SQL 语句长时间运行，我们想马上终止它，可以利用 Ctrl+C 组合键终止正在运行的 SQL 语句。

5．格式化运行结果

当执行以下 SQL 语句的时候，SQL 语句的返回结果比较凌乱，列名和列的返回值不相匹配，不易阅读，给大家带来了诸多的不便！如下所示：

```
SQL> SELECT log_id,src_addr,dest_addr,recv_time FROM processing_logs_arch
where rownum<5;
```

SQL 语句的执行结果是：

```
LOG_ID SRC_ADDR    DEST_ADDR
------------------------------------------------------------------------
RECV_TIME
------------------------------------------------------------------------
      268 13905710000 13805866601
06-2月 -05 12.02.35.777000 下午
      269 13905710000 13858270084
06-2月 -05 12.02.36.956000 下午
      270 13905710000 13566366427
06-2月 -05 12.02.38.835000 下午
   LOG_ID SRC_ADDR    DEST_ADDR
---------- ----------- -----------
RECV_TIME
------------------------------------------------------------------------
      271 13905710000 13805866601
06-2月 -05 12.02.38.940000 下午
     ……
```

SQL*Plus 不像其他第三方工具，它不能对 SQL 语句的返回结果自动进行格式化。我们可以使用 SQL*Plus 的 COLUMN 命令对报表进行格式化。

从上面的输出可以看出，RECV_TIME 列占用了太多的空间，我们可以缩小列 RECV_TIME 显示的宽度。

```
SQL> COLUMN recv_time FORMAT A40;
SQL> SELECT log_id,src_addr,dest_addr,recv_time FROM processing_logs_arch
```

```
where rownum<5;
```

SQL 语句的执行结果是：

```
LOG_ID    SRC_ADDR    DEST_ADDR  RECV_TIME
---------- ---------- --------- ---------------------------------------
   268    13905710000 13805866601 06-2月 -05 12.02.35.777000 下午
   269    13905710000 13858270084 06-2月 -05 12.02.36.956000 下午
   270    13905710000 13566366427 06-2月 -05 12.02.38.835000 下午
   271    13905710000 13805866601 06-2月 -05 12.02.38.940000 下午
```

对列进行格式化以后，输出结果就比较漂亮。

COLUMN recv_time FORMAT A40 是把 recv_time 列的显示宽度限制为 40 个字符。

其中，COLUMN 是 SQL*Plus 的专有命令，recv_time 是要格式化的列，FORMAT 也是关键字，表示要对列的输出进行格式化，A40 表示列的输出被限制为 40 个字符，A 表示字符。

COLUMN 用法的其他例子如下：

- `COLUMN LAST_NAME HEADING 'Family Name'` 格式化列 LAST_NAME，列的标题变成'Family Name'
- `COLUMN SALARY FORMAT $99,999` 把工资（SALARY）的显示格式化成$99,999 的形式，9 代表数字。格式化以后的工资将如下所示：

```
$17,000
$17,000
$14,000
$13,500
```

6．在 SQL*Plus 中执行存储过程

命令：EXECUTE

语法：EXEC[UTE] statement

用法：执行存储过程。

例子：`SQL>EXECUTE add_employee('kss',250);`

其中 add_employee 是存储过程的名称；“kss”、“250”是传入存储过程的两个参数。

7．修改 SQL*Plus 的启动选项

当启动 SQL*Plus 的时候，SQL*Plus 会自动读取站点配置文件 glogin.sql。这个文件存放 SQL*Plus 的各项设置。如果我们要更改 SQL*Plus 的启动选项，只需修改这个文件即可。这里所说的启动选项，其实就是 SQL*Plus 的环境变量。

STEP 01 直接修改文件 glogin.sql。

STEP 02 重新启动 SQL*Plus，使上一步对 SQL*Plus 选项的修改立即生效。

知识点索引——站点配置文件（Site Profile）

站点配置文件（Site Profile）是一个脚本文件，它控制使用 SQL*Plus 登录到数据

库的会话（session）的行为，当用户启动 SQL*Plus 登录数据库时，Oracle 会自动执行这个脚本文件，以设置好用户在 SQL*Plus 中的工作环境。

站点配置文件的名字通常叫 glogin.sql。这个文件的默认位置是：

✓ $ORACLE_HOME/sqlplus/admin/glogin.sql（UNIX 平台）

✓ %ORACLE_HOME%\sqlplus\admin\glogin.sql（Windows 平台）

站点配置文件是在安装 SQL*Plus 时创建的。如果卸载 SQL*Plus，站点配置文件也将被一同删除。

站点配置文件针对所有的会话，并不是针对特定的会话。站点配置文件使 DBA 能够为登录到数据库中的所有用户设置相同的 SQL*Plus 工作环境。



知识点索引——用户配置文件（User Profile）

用户配置文件（User Profile）也是一个脚本文件。当用户启动 SQL*Plus 登录数据库时，Oracle 首先自动执行站点配置文件（Site Profile），然后才自动执行用户配置文件（User Profile）。用户配置文件（User Profile）用于设定特定会话（session）的工作环境。

用户配置文件（User Profile）的名字是 login.sql。启动 SQL*Plus 登录数据库时，Oracle 首先在当前路径查找用户配置文件（User Profile），如果找不到，Oracle 会到操作系统环境变量 SQLPATH 指定的路径下进行查找。

8．重复执行一条 SQL 语句

如果想重复执行一条 SQL 语句，而又不想重复输入，SQL*Plus 提供了这样的功能，它允许用户重复执行最后一条 SQL 语句。SQL*Plus 提供了一个 SQL 缓冲区，当用户执行完一条 SQL 语句的时候，这条语句会被缓存在 SQL 缓冲区中，直到输入新的语句，覆盖原先的语句，或者用户退出 SQL*Plus。

在 SQL*Plus 中输入斜杠（/）将重复执行一条 SQL 语句。RUN 命令也可以用来重复执行一条 SQL 语句，它的功能和斜杠（/）相同。

STEP 01 执行一条 SQL 语句，这条语句将被缓存在 SQL 缓冲区中。

```
SQL> SELECT sysdate FROM dual;
```

SQL 语句的执行结果是：

```
SYSDATE
--------------
03-5月 -06
```

STEP 02 通过 LIST 命令查看缓冲区中的 SQL 语句，这条语句正是我们刚才执

行的语句。

```
SQL> LIST
  1* SELECT sysdate FROM dual
```

STEP 03 输入斜杠（/），将重复执行缓冲区的 SQL 语句。

```
SQL> /
SYSDATE
--------------
03-5月 -06
```

9．把查询结果输出到文件中

为了产生报表，我们需要将查询的结果保存到一个操作系统文件中，然后，我们可以对这个文件进行编辑、打印、合并，以生成我们需要的报表。有时候我们生成的查询结果就是一系列的 SQL 语句，我们可以将这些结果保存到文件中，在需要时，用 SQL*Plus 的@命令运行这个脚本文件。SQL*Plus 的 SPOOL 命令可以用来把查询结果保存到操作系统文件中。

将查询结果保存到文件中的过程说明如下。

STEP 01 指定将查询结果输出到哪个文件中。

```
SQL> SPOOL D:\script\sptest.txt
```

在本例中，将查询的结果集保存到文件 sptest.txt 中。

从这里开始，执行的每一句 SQL 语句及其结果将被保存到文件 sptest.txt 中。

STEP 02 执行各种各样的 SQL 语句（如下所示），结果集都会被保存到文件 sptest.txt 中。

```
SQL> SELECT * FROM student;
```

SQL 语句的执行结果是：

```
NAME                                     TELEPHONE
---------------------------------------- --------------------
小王                                     13913307891
小张                                     13913307891
小李                                     13913307891
小不点                                   13913307891
mary                                     13913307891
```

```
SQL> SELECT count(*) FROM A4;

  COUNT(*)
----------
      1000
```

执行 SPOOL off 命令，该命令用来终止向文件 sptest.txt 中写入查询的结果。也就是说，从这里开始，不再向 sptest.txt 中写入任何东西。

```
SQL> SPOOL off;
```

STEP 03 进入操作系统，查看文件 d:\script\sptest.txt 的内容。这个文件的内容如下所示：

```
SQL> SELECT * FROM student;
NAME                                     TELEPHONE
---------------------------------------- --------------------
小王                                     13913307891
小张                                     13913307891
小李                                     13913307891
小不点                                   13913307891
mary                                     13913307891
SQL> SELECT count(*) FROM A4;
  COUNT(*)
----------
      1000
SQL> SPOOL off;
```

对文件进行编辑，便可获得我们需要的报表或结果。

10．在 SQL 语句中使用替代变量（Substitution Variable）

在 SQL*Plus 执行 SQL 语句或者 SQL*Plus 命令，如果想交互输入值，需要使用替代变量。替代变量以**&**开始。当 SQL*Plus 遇到**&**时，会自动提示用户输入值。如下例所示，我们可以动态输入手机的单价（PRICE）。

```
SQL> SELECT * FROM PHONE_INFOR WHERE PRICE>&unitprice;
输入 unitprice 的值:  500
原值    1: SELECT * FROM PHONE_INFOR WHERE PRICE>&unitprice
新值    1: SELECT * FROM PHONE_INFOR WHERE PRICE>500
PHONE_NAME          PRODUCER                PRICE
------------------- ------------------- ----------
N97                    诺基亚                  700
C510c                  爱立信                  1201
```

每次执行这条 SQL 语句，不需要对 SQL 语句做任何修改，只需按照提示输入值即可。



5.2 Web 版的企业管理器（Oracle Enterprise Manager，OEM）

Oracle 企业管理器（OEM）是一款图形界面管理工具，用于对数据库进行管理，特别适合对命令行不熟悉的使用者。

以前的 Oracle 企业管理器是一款独立的软件，现在被 Oracle 公司做成了 Web 形式的，与 Oracle 数据库服务器捆绑在一起，我们把它称为“Web 版的企业管理器”。只要有浏览

器，我们就可以通过它进行数据库管理。

要想使用 Web 版的企业管理器，数据库服务器端必须存在 Oracle 企业管理器数据库控件（在此，我们简称“数据库控件”）。数据库创建时，我们可以选择是否安装它。

企业管理器可以执行下面的管理任务：

- 管理模式对象（如创建表、视图、索引等）
- 管理用户
- 管理内存
- 执行备份/恢复
- 导入/导出数据
- 查看数据库的性能和状态信息

5.2.1 使用 OEM

上面我们提到，OEM 是一款基于 Web 的管理工具，接下来，我们将演示如何利用这个工具来管理数据库。需要注意的是，这个工具实现的功能和 SQL*Plus 实现的功能大部分相同。该工具直观、方便，但是功能上没有 SQL*Plus 那么强大。Web 版的企业管理器的使用步骤说明如下。

STEP 01 必须确保进程 dbconsole 正在运行。

知识点索引——dbconsole 进程

客户端浏览器为了能够访问 Oracle 企业管理器数据库控件（Oracle Enterprise Manager Database Control），需要数据库控件控制台进程（Database Control Console Process）dbconsole 的支持，该进程运行在数据库服务器端（如图 5-2 所示）。当数据库创建完成以后，该进程被自动启动。也可以手工启动该进程。

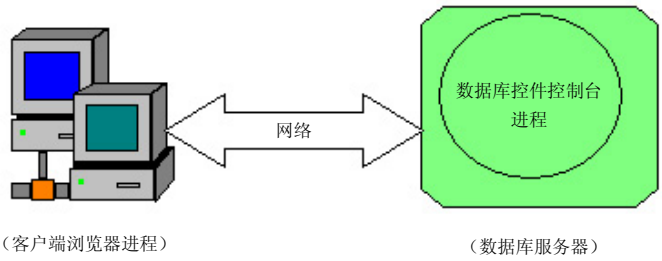


图 5-2 数据库控件控制台进程原理

要想使用 Web 版的企业管理器，dbconsole 进程必须在数据库服务器端运行。

为了验证进程 dbconsole 是否正在运行，请执行下面的步骤。

ⓐ 登录操作系统，并进入目录 Oracle_home/bin，其中 Oracle_home 表示 Oracle 主目录。

ⓑ 执行命令 emctl status dbconsole。

```
emctl status dbconsole
Oracle Enterprise Manager 11g Database Control Release 11.1.0.6.0
Copyright (c) 1996, 2007 Oracle Corporation.  All rights reserved.
https://zero:1158/em/console/aboutApplication
Oracle Enterprise Manager 11g is running.
```

从最后一行不难看出，dbconsole 正在运行。

STEP 02 在浏览器中输入地址，出现如图 5-3 所示的界面。

在浏览器中输入地址 https://zero:1158/em（地址的格式是 https://hostname:portnumber/em，其中 hostname 表示主机名，portnumber 表示端口号）。注意，这里使用的是 https 而不是 http。



图 5-3　登录

知识点索引——如何获知端口号

- ✓ 如果在 Linux 或者 UNIX 平台，请查看文件 Oracle_home/install/portlist.ini。
- ✓ 如果是 Windows 平台，单击“开始（Start）”，选择“所有程序（All Programs）”，单击“Oracle 主目录（HOME_NAME）”，把鼠标移动到“Database Control Entry”，单击鼠标右键，选择“属性”。

输入用户名 system，口令 zero，单击“登录”按钮，将登录到数据库中，并显示如图 5-4 所示的页面。



图 5-4　企业管理器主界面

该页面显示了数据库的状态等信息。在这个界面中，我们可以执行大量的数据库管理任务。

5.2.2　启动 dbconsole

如果我们不能够从 Web 页面管理远程数据库，很可能是数据库控件控制台进程

（dbconsole）没有启动。启动该进程的方法说明如下。

STEP 01 登录操作系统，并进入目录 Oracle_home/bin，其中 Oracle_home 表示 Oracle 主目录。

STEP 02 执行命令 emctl start dbconsole。

```
emctl start dbconsole
Oracle Enterprise Manager 11g Database Control Release 11.1.0.6.0
Copyright (c) 1996, 2007 Oracle Corporation.  All rights reserved.
https://zero:1158/em/console/aboutApplication
Starting Oracle Enterprise Manager 11g Database Control ...OracleDBConsoleZDB
服务正在启动 ..................................................
OracleDBConsoleZDB 服务已经启动成功。
```

5.2.3 关闭 dbconsole

某些时候，需要关闭进程 dbconsole。关闭 dbconsole 的过程说明如下。

STEP 01 登录操作系统，并进入目录 Oracle_home/bin，其中 Oracle_home 表示 Oracle 主目录。

STEP 02 执行命令 emctl stop dbconsole。

```
emctl stop dbconsole
Oracle Enterprise Manager 11g Database Control Release 11.1.0.6.0
Copyright (c) 1996, 2007 Oracle Corporation.  All rights reserved.
https://zero:1158/em/console/aboutApplication
OracleDBConsoleZDB 服务正在停止.....................
OracleDBConsoleZDB 服务已成功停止。
```

5.3 其他数据库访问工具

从关系数据库的发明到如今，几大数据库产品垄断了全球市场，关系数据库的天堂似乎一片宁静；而数据库客户端市场却风起云涌，你方唱罢，我方登场，真是代代皆有人才出。其中的佼佼者有 DbVisualizer、TOAD、Embarcadero SQL Tuner、PLSQL Developer……需要注意的是，以上几种工具大同小异，各有千秋！

5.3.1 DbVisualizer

DbVisualizer 是一个专为开发人员和数据库管理人员设计的工具，用于数据库的开发和管理。其跨平台性令人吃惊，可以运行在 Windows、Mac OS X、Linux/UNIX 上；它能访问 Oracle、Sybase、SQL Server、PostgreSQL、DB2、Mimer、Neoview、MySQL、Informix 等数据库系统。DbVisualizer 有以下几大功能：

- 数据库浏览（Database Browser）。
- 数据库对象管理（Database Object Management），创建、修改、删除数据库对象。
- 表数据的管理（Table Data Management），编辑数据（包括二进制、BLOB、CLOB），从文件导入数据。
- SQL 语句工具（SQL Tools），支持自动创建 SQL 语句，查看执行计划等。
- 数据库服务器管理（Database Server Management），管理数据库实例、存储、安全等。
- 工具（Tools），可以生成对象的 DDL 语句和表数据。



5.3.2 TOAD

TOAD（Tool of Oracle Application Developer）是一款专业化、图形化的应用程序开发和数据库管理工具，它具有简单易用、功能强大等特点。TOAD 具有功能强大的模式浏览器，可以快速访问数据字典、表、索引和存储过程等，并可以在窗口中直接操作数据库对象。内置的 SQL 编辑器可以生成、编辑、运行和优化 SQL 语句。TOAD 为 DBA 节省了大量的时间，提高了工作效率。多年来，TOAD 在 Oracle 专业人员中享有很高的盛誉（特别是 DBA），它已经成为许多 Oracle 专业人士的首选工具。



5.3.3 Embarcadero SQL Tuner

Embarcadero SQL Tuner 是一款功能非常强大的最佳化数据库软件，用于优化你的 SQL 语句，提高 SQL 的执行效率。它提供了友好的图像界面，快速帮助你侦测和调整效率低下的 SQL 语句。Embarcadero SQL Tuner 简化了数据库的管理，提高了数据库的运行效率。



5.3.4 PL/SQL Developer

PL/SQL Developer 是一款非常方便的 PL/SQL 程序编写与调试工具。它集成强大的开发环境，专门用于数据库的开发、调试等。PL/SQL Developer 是数据库管理与使用的好帮手，甚至连 Oracle 公司内部的人员都在使用它。PL/SQL Developer 功能十分全面，大大缩短了软件开发的生命周期。

接下来我们将简单介绍 PL/SQL Developer 的使用。

1．用 PL/SQL Developer 登录远程数据库

在启动 PL/SQL Developer 前，必须在客户端配置好网络服务名，网络服务名用于使客户端能够连接到远程数据库。网络服务名的配置，**请参考本书 4.2.1 节的内容**。本文以 PL/SQL Developer 7 为例进行说明。

单击“开始”→“程序”→“PL/SQL Developer” →“PL/SQL Developer”出现如图 5-5 所示的界面。

- 在 Username 中，输入远程数据库用户的用户名（system）。

- 在 Password 中，输入远程数据库用户（system）的密码（zero）。
- 在 Database 中，输入网络服务名（以前创建的 MYZDB），网络服务名定义了要连接的远程数据库。
- 在 Connect as 下拉列表中指定连接方式，有三种选择：Normal（以普通身份登录数据库）、SYSDBA（以 SYSDBA 身份登录数据库）、SYSOPER（以 SYSOPER 身份登录数据库）。

登录以后的界面如图 5-6 所示。

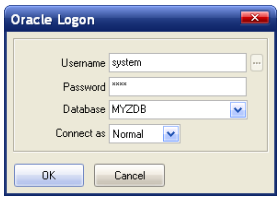


图 5-5　登录

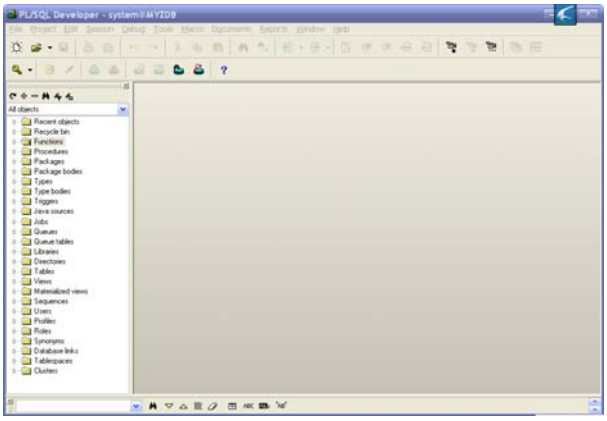


图 5-6　PL/SQL Developer 主界面

2．PL/SQL Developer 的常用功能

在 PL/SQL Developer 中，可以浏览对象、创建对象、修改对象、导入数据、导出数据、生成创建表的 SQL 语句，还可以在 PL/SQL Developer 中执行 SQL 命令。总之，PL/SQL Developer 功能强大。由于篇幅宝贵，本书只摘出 PL/SQL Developer 的两个例子。

示例 1　浏览数据库对象

图 5-7 列出了各种数据库对象的分类，如函数（Functions）、存储过程（Procedures）、表（Tables）、视图（Views）等。

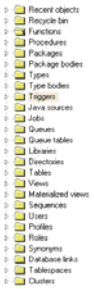


图 5-7　对象分类

展开“Tables”，如图 5-8 所示，将会列出登录用户所拥有的表。

示例 2　创建对象

STEP 01 在 Tables 上单击鼠标右键，选择“New”命令，出现如图 5-9 所示的界面。

指定要创建的表的名字（本例中是 firsttable），指定该表存放的表空间（本例中是 myspace），单击 Columns 标签，出现如图 5-10 所示的界面。

STEP 02 指定列的属性。



图 5-8　用户的表



图 5-9　指定表的属性



图 5-10　指定列的属性

在该窗口中，指定列的属性。本例中列 id 的数据类型是 number，列 name 的数据类型是 varchar2(30)，列长度是 30。单击“Apply”按钮及“Close”按钮，表 firsttable 创建成功。



本章总结

本章介绍了用于访问数据库的各种工具。如果你是 DBA，推荐使用 SQL*Plus，它能让你自如地管理数据库，同时也要求你有较高的数据库管理技巧。一旦你熟悉命令行，就会发觉 SQL*Plus 的使用其实没有那么吃力。如果你是初级 DBA，Web 版的企业管理器能够帮助你轻易完成许多任务，但从长远来说，不要过分依赖该工具。其他诸如 DbVisualizer、TOAD、Embarcadero SQL Tuner、PL/SQL Developer 等工具，比较适合开发人员使用，不过，有些功能特别适合 DBA 使用。

第 6 章　Oracle 数据库启动/关闭——原理剖析

数据库启动、关闭的过程其实挺简单，但是原理却比较复杂。本章的重点是理解 Oracle 启动和关闭的原理。

6.1　数据库的启动

数据库的启动（STARTUP）/关闭（SHUTDOWN）是最容易出现故障的地方。启动/关闭数据库的命令很简单，但是启动的过程中 Oracle 做了哪些工作、关闭的过程中 Oracle 又做了哪些工作，这需要我们深刻理解 Oracle 启动/关闭的原理。这就是本章学习的重点。

6.1.1　启动一个关闭的数据库（常规启动）

数据库的常规启动过程说明如下。

STEP 01 用命令 SQLPLUS /NOLOG 打开 SQL*Plus。

```
C:\Documents and Settings\Administrator>SQLPLUS /NOLOG
SQL*Plus: Release 11.1.0.6.0 - Production on 星期五 7 月 24 17:43:52 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
```

STEP 02 以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
输入口令:
已连接。
```

知识点索引——系统权限 SYSDBA、SYSOPER

SYSDBA 和 SYSOPER 是数据库中的两个超级权限，同属于系统权限（System Privileges），Oracle 允许在数据库没有打开的情况下，拥有这两个权限的用户访问实例。在执行重大的数据库操作（尤其是数据库的启动和关闭）时，需要这两个权限。

SYSDBA 可以执行下面的任务：

- ✓ 启动、关闭数据库（STARTUP、SHUTDOWN）
- ✓ 修改数据库（ALTER DATABASE），包括打开、装载、备份、改变字符集等
- ✓ 创建数据库（CREATE DATABASE）
- ✓ 删除数据库（DROP DATABASE）
- ✓ 创建服务器参数文件（CREATE SPFILE）
- ✓ 改变归档模式（ALTER DATABASE ARCHIVELOG）
- ✓ 执行数据库恢复（ALTER DATABASE RECOVER）

SYSOPER 的权限比 SYSDBA 的小。

STEP 03 用 STARTUP 命令启动数据库，如图 6-1 所示。

从图 6-1 中可以看出，数据库的启动经历了 3 个阶段：启动实例、装载数据库、打开数据库。

当命令 STARTUP 执行完毕后，我们也可以从警报文件（Alert Log）中获得数据库启动的详细过程（如图 6-2 所示）。

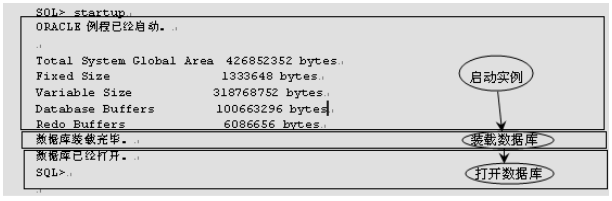


图 6-1　数据库启动

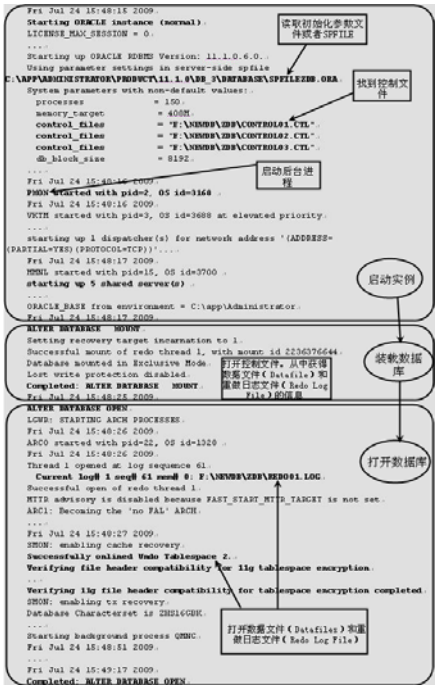


图 6-2　数据库启动过程（来源于警报文件）

知识点索引——查找警报文件（Alert Log）

ⓐ 登录数据库。

ⓑ 找出警报文件的路径。

用 SHOW PARAMETER 命令显示出警报文件的路径。

```
SQL> SHOW PARAMETER BACKGROUND_DUMP_DEST
NAME                 TYPE        VALUE
-------------------------------- ----------- ------------------------------
background_dump_dest  string    c:\app\administrator\diag\rdbms\zdb\zdb\trace
```

找到警报文件。

警报文件的文件名格式通常是 alert_instance.log，本例中警报文件的名字是 alert_zdb.log。

参考

有关警报文件的详细介绍，**请参考本书 6.2 节的内容**。

6.1.2 数据库启动原理

对于 DBA 或者想深入学习 Oracle 的朋友来说，深刻理解 Oracle 的启动过程尤为必要。数据库的启动原理是 Oracle 最基础的部分，也是最重要的部分。深刻理解数据库的启动机制，可以帮助我们排查启动过程中的疑难杂症。要想深刻理解 Oracle 的启动过程，我们需要了解启动阶段涉及哪些文件与哪些进程。

Oracle 数据库的启动要经历 3 个阶段：启动实例、装载数据库、打开数据库（如图 6-3 所示）。



图 6-3　数据库启动的三个阶段

STARTUP 命令一次完成了数据库启动的 3 个阶段。当然，我们也可以一步一步手动启动数据库，或者直接把数据库启动到某个特定的阶段（后续会有介绍）。

1．启动实例（Start An Instance）

参考图 6-1 与图 6-2，当启动一个实例时，Oracle 会**读取**一个参数文件，这个文件可以是初始化参数文件（Initialization Parameter File，PFILE），也可以是服务器端参数文件（Server Parameter Files，SPFILE），通常，我们把两者都称为**参数文件（Parameter File）**，Oracle 根据参数文件（PFILE、SPFILE）中的参数，分配相应大小的一片内存区域叫系统全局区（System Global Area，SGA），然后启动一系列的后台进程（DBWR：数据库写入进程、LGWR：日志写入程序、CKPT：检查点进程、SMON：系统监控进程、PMON：进程监控进程、ARCH 归档进程等）。这些内存和进程合起来组成**实例（Oracle Instance）**。Oracle 还依据参数文件（PFILE、SPFILE）进行其他一些设置（有关初始化参数的介绍，

请参考本书 6.1.3 节的内容。

关于实例的概念，对于初学 Oracle 的朋友来说，最不好理解。我们只需记住，Oracle 的内存和进程构成了实例。一台机器上可以同时运行多个实例。每个实例都有自己的名字（SID）。实例是用来驱动数据库的，在 RAC（Real Application Cluster，集群）环境中，多个实例可以同时驱动一个数据库。实例启动完成以后，数据库就处于 NOMOUNT 状态。

在实例启动完成以后，数据库还没有和实例关联。这时候，数据库不可访问。这个阶段主要用于数据库的维护（如重建控制文件等）。

参数文件（PFILE 或者 SPFILE）指定了控制文件（Control File）的位置。

值得一提的是，在创建数据库的时候，我们设置了初始创建的服务器进程数为 5（**请参考本书 2.2.3 节的内容**，在启动实例的时候，从图 6-2 的“**starting up 5 shared server(s)**”中不难看出，Oracle 启动了 5 个服务器进程。

2．装载数据库（Mount The Database）

参考图 6-1 与图 6-2，在这个阶段中，Oracle 根据参数文件（PFILE 或者 SPFILE）中的参数（CONTROL_FILES）找到控制文件（Control File），然后打开控制文件。从控制文件中获得数据文件（Datafile）和重做日志文件（Redo Log File）的名字及位置。这个时候，Oracle 已经把实例（Instance）和数据库关联起来。对于普通用户来说，数据库还是不可访问。

处于 MOUNT 阶段的数据库，主要用于数据库的维护（如恢复数据库等）。

3．打开数据库（Open The Database）

参考图 6-1 与图 6-2，当打开数据库的时候，Oracle 打开数据文件（Datafiles）和重做日志文件（Redo Log File）。这时候，数据库可以使用。普通用户可以登录数据库，并对数据库进行操作。



6.1.3 初始化参数（Initialization Parameter）

PFILE 和 SPFILE 是两种类型的参数文件，它们都用于启动数据库。启动数据库时，Oracle 会读取这两个文件中的信息（Oracle 一次只使用一个），并进行相关的设置（如设置高速缓冲区的大小、指定控制文件的位置等）。PFILE 是文本文件，可以使用编辑器修改该文件的内容。SPFILE 是二进制文件，不能使用编辑器直接修改它的内容，SPFILE 的内容要通过 Oracle 的命令进行读写。在实例运行时，DBA 可以动态修改 SPFILE 中的参数，因此，Oracle 推荐使用 SPFILE。

数据库中有许多**初始化参数（Initialization Parameter）**，这些参数用于对整个数据库进行限制、对用户和进程进行限制、对数据库资源进行限制、对数据库的性能进行调整。部分初始化参数如下所示：

```
DB_BLOCK_SIZE
DB_NAME
```

```
SESSIONS
CLUSTER_DATABASE
COMPATIBLE
CONTROL_FILES
DB_CREATE_FILE_DEST
DB_CREATE_ONLINE_LOG_DEST_n
DB_DOMAIN
DB_RECOVERY_FILE_DEST
DB_RECOVERY_FILE_DEST_SIZE
DB_UNIQUE_NAME
INSTANCE_NUMBER
JOB_QUEUE_PROCESSES
LOG_ARCHIVE_DEST_n
LOG_ARCHIVE_DEST_STATE_n
NLS_LANGUAGE
NLS_TERRITORY
OPEN_CURSORS
PGA_AGGREGATE_TARGET
PROCESSES
REMOTE_LISTENER
REMOTE_LOGIN_PASSWORDFILE
ROLLBACK_SEGMENTS
SGA_TARGET
SHARED_SERVERS
STAR_TRANSFORMATION_ENABLED
UNDO_MANAGEMENT
UNDO_TABLESPACE
……
```

Oracle 把上面的初始化参数及其值存放在一个文件中，我们把这个文件叫**参数文件**。启动数据库时（严格地说，应该叫启动实例），Oracle 会读取**参数文件**中的值，然后根据这些值对实例和数据库进行设置。

参数文件（Parameter File）分成两类：

- 初始化参数文件（Initialization Parameter Files，PFILE），通常简称为“PFILE”；
- 服务器参数文件（Server Parameter Files，SPFILE），通常简称为“SPFILE”。

PFILE、SPFILE 内容相同，只是文件的格式不同而已。一个（PFILE）是文本文件，一个（SPFILE）是二进制文件。

1．PFILE

PFILE 是传统的参数文件。它是一个文本文件，用于存放初始化参数。一个初始化参数文件的样例如下所示：

```
db_name='ZDB'
java_pool_size=4194304
job_queue_processes=10
undo_management='AUTO'
```

```
undo_tablespace='UNDOTBS1'
user_dump_dest='D:\ORACLE\PRODUCT\10.2.0\ADMIN\ZDB\UDUMP'
log_buffer=11154944
lock_sga=TRUE
……
```

参数文件的缺点是，在数据库运行时，DBA 不能永久修改初始化参数的值。要想永久修改初始化参数的值，DBA 需要先关闭数据库，对参数文件（PFILE）进行修改，然后用修改后的 PFILE 重新启动数据库。

2．SPFILE

SPFILE 是一个位于数据库服务器上的二进制文件。和初始化参数文件（PFILE）一样，服务器参数文件也用于存放初始化参数，不同的是，在数据库运行的时候，DBA 可以永久修改初始化参数的值。

服务器参数文件的默认名字及默认位置如表 6-1 所示。

表 6-1　服务器参数文件的默认名字及默认位置

操作系统	默认名字	默认路径
UNIX/Linux	spfile$ORACLE_SID.ora	$ORACLE_HOME/dbs
Windows	spfile$ORACLE_SID.ora	%ORACLE_HOME\database

3．显示初始化参数的值

要想查看初始化参数的设置，有两种方法：用 SHOW 命令和查询视图 V$PARAMETER。

■ 显示所有初始化参数的值。

➢ 方法 1：

```
SELECT name,value
FROM V$PARAMETER
```

其中，name 是参数的名字；value 是参数的值。

➢ 方法 2：

```
SQL> SHOW PARAMETERS
```

其中，PARAMETERS 是关键字，表示显示所有参数的值。

■ 显示指定初始化参数的值。

➢ 方法 1：

```
SQL> SHOW PARAMETER undo_retention
```

SQL 语句的执行结果是：

```
NAME                                 TYPE            VALUE
------------------------------------ ----------- -----
undo_retention                       integer         900
SQL>
```

➢ 方法 2：

```
SQL> SELECT NAME,VALUE
  2  FROM V$PARAMETER
  3  WHERE name='undo_retention '
  4  /
```

SQL 语句的执行结果是：

```
NAME                      VALUE
------------------ -------------
undo_retention     900
SQL>
```

从输出可以看出，初始化参数 undo_retention 的值是 900

4．初始化参数的修改

为了对数据库进行限制或者调整数据库的性能，DBA 需要**永久**修改初始化参数，则：

➢ 如果数据库使用的是初始化参数文件（PFILE），要想**永久**修改初始化参数文件中的参数，DBA 需要经历关闭数据库→修改初始化参数文件（PFILE）中的参数→重新启动数据库的过程。

➢ 如果数据库使用的是服务器参数文件（SPFILE），DBA 只需使用 ALTER SYSTEM SET ... SCOPE=SPFILE 命令修改初始化参数，这个命令会更新初始化参数文件（SPFILE）。

如果 DBA 只是想临时更改初始化参数的值，可以使用 ALTER SYSTEM SET ... SCOPE=MEMORY 命令，这种更改只对当前实例有效，也就是说，如果 DBA 重新启动实例，刚才的修改将会丢失。用户也可以使用 ALTER SESSION 命令改变某些初始化参数的值，但是这种改变仅限于本次会话。

5．PFILE、SPFILE 的相互转换

■ 根据 SPFILE 创建 PFILE

为了备份初始化参数的设置，可以根据 SPFILE 生成 PFILE（一个文本文件）。PFILE 也可以用来启动数据库。根据 SPFILE 创建 PFILE 的语法如下所示：

```
CREATE PFILE = 'PFILE 的名字' FROM SPFILE= 'SPFILE 的名字'
```

要想执行这个命令，用户必须拥有 SYSDBA 或者 SYSOPER 权限。创建 PFILE 的过程说明如下。



以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
Enter password:
Connected.
```



创建 PFILE。



注：本例中的 SPFILE(SPFILEZDB.ORA)是在创建数据库时由 Oracle 自动创建的。

执行下面的语句创建 PFILE：

```
    SQL> CREATE PFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
      2  FROM  SPFILE = 'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
      3 /
```

文件已创建。

- C:\app\Administrator\product\11.1.0\db_3\database\INITZDB.ORA 是 PFILE 的名字及路径。



注：PFILE 可以为任意名字，本例中的名字是 INITZDB.ORA。

- C:\app\Administrator\product\11.1.0\db_3\database\SPFILEZDB.ORA 是 SPFILE 的名字及路径。

STEP 03 到操作系统中检查新创建 PFILE 的存在性。

■ 根据 PFILE 创建 SPFILE

如果 SPFILE 丢失，我们可以根据 PFILE 创建 SPFILE。要想使用 SPFILE 启动数据库，必须提前创建 PFILE。创建 SPFILE 的语法如下所示：

```
    CREATE SPFILE = 'SPFILE 的名字' FROM PFILE= 'PFILE 的名字'
```

要想执行这个命令，用户必须要有 SYSDBA 或者 SYSOPER 权限。

STEP 01 以 SYSDBA 的身份登录数据库。

```
    SQL> CONNECT sys AS SYSDBA
    Enter password:
    Connected.
```

STEP 02 关闭数据库。

```
    SQL> SHUTDOWN IMMEDIATE;
```

数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。

STEP 03 创建 SPFILE。



注：为了测试方便，我们将已经存在的 SPFILE 的名字由 SPFILEZDB.ORA 改成 SPFILEZDB2.ORA。

```
    SQL> CREATE SPFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
    2    FROM  PFILE  =  'C:\app\Administrator\product\11.1.0\db_3\database\
```

```
INITZDB.ORA'
    3  /
    文件已创建
```

- C:\app\Administrator\product\11.1.0\db_3\database\SPFILEZDB.ORA 是 SPFILE 的名字及路径。通常，SPFILE 的名字要满足这样（SPFILE$ORACLE_SID.ORA）的命名规范。
- C:\app\Administrator\product\11.1.0\db_3\database\INITZDB.ORA 是 PFILE 的名字及路径。

STEP 04 到操作系统中检查新创建 SPFILE 的存在性。

从操作系统中可以看出，已经创建文件 SPFILEZDB.ORA。

STEP 05 重新启动数据库。

```
SQL> STARTUP
ORACLE 例程已经启动。
Total System Global Area  426852352 bytes
Fixed Size                 1333648 bytes
Variable Size            314574448 bytes
Database Buffers         104857600 bytes
Redo Buffers               6086656 bytes
数据库装载完毕。
数据库已经打开。
```

Oracle 使用服务器参数文件（SPFILE）SPFILEZDB.ORA 启动数据库。

6．使用参数文件（PFILE、SPFILE）启动数据库

DBA 可以使用 SPFILE 启动数据库，也可以使用 PFILE 启动数据库。使用 SPFILE 启动数据库，只需输入 STARTUP 命令，无须指定其他选项；使用 PFILE 启动数据库，需要在 STARTUP 命令的后面指定参数 PFILE，PFILE 参数指定初始化参数文件（PFILE）的名字及位置。

■ 使用 SPFILE 启动数据库

使用 SPFILE 启动数据库的过程说明如下。

STEP 01 以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
Enter password:
Connected.
```

STEP 02 使用 SPFILE 启动数据库。

```
SQL> STARTUP
ORACLE 例程已经启动。
Total System Global Area  426852352 bytes
Fixed Size                 1333648 bytes
Variable Size            314574448 bytes
Database Buffers         104857600 bytes
```

```
Redo Buffers              6086656 bytes
```
数据库装载完毕。
数据库已经打开。

在这里，虽然命令行中没有指定 SPFILE 的名字及位置，但是，Oracle 会自动查找 SPFILE，然后用它来启动数据库。Oracle 要根据一定规则查找 SPFILE 文件，这就是为什么 SPFILE 的命名一定要规范的原因；如果 SPFILE 的命名不规范，即使 SPFILE 确实存在，Oracle 也找不到该文件。

■ 使用 PFILE 启动数据库

使用 PFILE 启动数据库的过程说明如下。

STEP 01 以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
Enter password:
Connected.
```

STEP 02 使用 PFILE 启动数据库。

```
SQL>  STARTUP  PFILE='C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
ORACLE 例程已经启动。
Total System Global Area  426852352 bytes
Fixed Size                 1333648 bytes
Variable Size            314574448 bytes
Database Buffers         104857600 bytes
Redo Buffers               6086656 bytes
数据库装载完毕。
数据库已经打开。
SQL>
```

Oracle 使用 PFILE 启动数据库，在命令行中需要指定 PFILE 的名字及位置，在本例中，PFILE 的名字和位置是 C:\app\Administrator\product\11.1.0\db_3\database\INITZDB.ORA。



注：如果在 STARTUP 命令之后不指定 PFILE ,Oracle 将使用 SPFILE 启动数据库。



6.1.4 只启动实例——把数据库启动到非装载状态（NOMOUNT）

进行数据库维护时，需要把数据库启动到 NOMOUNT 状态。在执行 STARTUP 命令时，如果在其之后指定选项 NOMOUNT，将把数据库启动到 NOMOUNT 状态，也就是只启动实例，详细过程说明如下。

STEP 01 启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```

STEP 02 启动数据库到 NOMOUNT 状态。

```
SQL>  STARTUP NOMOUNT
ORACLE 例程已经启动。
Total System Global Area  426852352 bytes
Fixed Size                 1333648 bytes
Variable Size            314574448 bytes
Database Buffers         104857600 bytes
Redo Buffers               6086656 bytes
SQL>
```

启动实例包括分配内存和启动进程，从上面的启动信息可以看出内存的分配信息。但是，并未出现信息“数据库装载完毕。数据库已经打开”，说明数据库只启动到“NOMOUNT”状态，也就是仅仅启动了实例（仅完成数据库启动的第一个阶段）。



6.1.5 把数据库启动到装载状态（MOUNT）

MOUNT 状态是数据库启动的第二个阶段。在启动命令 STARTUP 之后指定启动选项 MOUNT，能够把数据库启动到 MOUNT 状态，MOUNT 状态不仅启动了实例，还打开了控制文件，从控制文件中读取了数据文件和重做日志文件的名称和位置信息。

把数据库启动到 MOUNT 状态的过程还包括把数据库启动到 NOMOUNT 的过程。

■ 如果数据库是关闭的，则执行下面的过程把数据库启动到 MOUNT 状态。


启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```


启动数据库到 MOUNT 状态。

```
SQL> STARTUP MOUNT
ORACLE 例程已经启动。
Total System Global Area  426852352 bytes
Fixed Size                 1333648 bytes
Variable Size            314574448 bytes
Database Buffers         104857600 bytes
Redo Buffers               6086656 bytes
数据库装载完毕。
SQL>
```


从上面的信息容易看出，上述命令不仅启动了实例，还装载了数据库。但是，没有出现信息“数据库已经打开”，说明数据库还没有被打开，数据库的启动只是完成了前面的两个阶段。

■ 如果数据库处于 NOMOUNT 状态，则执行下面的命令把数据库启动到 MOUNT 状态。

```
SQL> ALTER DATABASE MOUNT
  2  /
Database altered.
```

ALTER DATABASE MOUNT 命令使数据库的状态从 NOMOUNT 变到 MOUNT。

6.1.6 打开数据库

打开数据库，使数据库可用。数据库所处的阶段不同，打开数据库的方法就不同。

■ **方法 1**：如果数据库是关闭的，则执行下面的过程。

STEP 01 启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```

STEP 02 启动数据库。

```
SQL> STARTUP
ORACLE 例程已经启动。
Total System Global Area  426852352 bytes
Fixed Size                 1333648 bytes
Variable Size            314574448 bytes
Database Buffers         104857600 bytes
Redo Buffers               6086656 bytes
数据库装载完毕。
数据库已经打开。
SQL>
```

这个过程包括把数据库启动到 NOMOUNT 状态，然后启动到 MOUNT 状态，最后打开数据库。

从启动信息可以看出，数据库启动经历了 3 个阶段。

阶段 1：NOMOUNT（分配内存）。

```
Total System Global Area  426852352 bytes
Fixed Size                 1333648 bytes
…
```

阶段 2：MOUNT（打开控制文件）。

```
"数据库装载完毕"
```

阶段 3：打开数据库。

```
"数据库已经打开"
```

■ **方法 2**：如果数据库处于 NOMOUNT 状态，则要执行下面的过程。

STEP 01 把数据库启动到 MOUNT 状态。

```
SQL> ALTER DATABASE MOUNT
  2  /
Database altered.
```

STEP 02 打开数据库。

```
SQL> ALTER DATABASE OPEN;
Database altered.
SQL>
```

■ **方法 3**：如果数据库处于 MOUNT 状态，则执行下面的过程直接打开数据库。

```
SQL> ALTER DATABASE OPEN;
Database altered.
```

6.1.7 重启你的数据库（Reset）

如果要重新启动一个数据库，我们应该首先关闭数据库，然后再启动数据库，这个过程相当于 Windows 中的“Reset（复位）”。在 Oracle 中，“Reset”的过程是可以通过一个启动选项（FORCE）来完成的。STARTUP FORCE 命令先调用 STARTUP ABORT 来关闭数据库，然后再用 STARTUP 命令，按照正常方式启动数据库。简而言之，STARTUP FORCE 命令就是把数据库的“关闭”和“启动”两个过程合二为一。

STEP 01 启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```

STEP 02 重启数据库。

```
SQL> STARTUP FORCE;
```

在启动数据库时，指定启动选项 FORCE，表示重新启动数据库。

STARTUP FORCE 命令会执行实例（Instance）恢复的过程，因此，在系统运行正常的情况下，不推荐使用这个命令，我们还是应该按照正常的顺序启动/关闭数据库。

6.1.8 把数据库启动到只读模式

在只读模式（Read Only）下，数据文件和重做日志文件都禁止写操作。整个数据库是只读的。不过，在只读模式下，允许执行数据库恢复（Database Recovery）及其他不会产生回滚的操作。

STEP 01 启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```

STEP 02 以只读方式启动数据库。

```
SQL> STARTUP OPEN READ ONLY;
```

STEP 03 验证数据库是否以只读方式打开。

```
SQL> SELECT open_mode FROM v$database;
OPEN_MODE
----------
READ ONLY
```

查询显示，数据库以只读方式打开。

6.1.9 把数据库设置成自动启动/关闭之——Windows 篇

无论是在 UNIX 平台还是 Windows 平台，都可以把 Oracle 设置成随操作系统一起启

动/关闭。在 Windows 平台，只需把 Oracle 的服务（Service）设置成自动启动和修改注册表，便可使 Oracle 随操作系统一起启动/关闭。Windows 上的设置要比 UNIX 上的设置简单得多。

STEP 01 把 Oracle 的服务设置成自动启动。

ⓐ 单击“开始”→“控制面板”→“性能和维护”→“管理工具”→“服务”。

ⓑ 打开 Oracle 的服务，将 Oracle 的服务设置成自动启动，如图 6-4 所示。



图 6-4　Oracle 的服务

Oracle 服务的格式是 OracleServiceSID，SID 表示 Oracle 的实例名（Instance）。本例中的服务名称是 OracleServiceZDB。

在“启动类型”中，选择“自动”，然后单击“确定”按钮。

STEP 02 编辑注册表。

ⓐ 单击“开始”→“运行”→输入命令 regedit 打开注册表。

ⓑ 进入 HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ORACLE_HOME_NAME。

ⓒ 把数据库设置成自动启动。

将参数 ORA_SID_AUTOSTART 设置为 TRUE。ORA_SID_AUTOSTART=TRUE 表示操作系统启动时，将自动启动 Oracle 数据库。

ⓓ 把数据库设置成自动关闭。

将参数 ORA_SID_SHUTDOWN 设置为 TRUE。ORA_SID_SHUTDOWN＝TRUE 表示操作系统关闭时，将自动关闭 Oracle 数据库。

将 ORA_SID_SHUTDOWNTYPE 设置成 IMMEDIATE。ORA_SID_SHUTDOWNTYPE 用于设置数据库的关闭方式。数据库的关闭方式有 ABORT、IMMEDIATE、NORMAL。IMMEDIATE 是最安全的数据库关闭方式。

STEP 03 创建 SPFILE（如果 SPFILE 不存在）。

操作系统启动时，将自动使用 SPFILE 启动数据库。这一点非常重要。所以，我们在设置数据的自动启动时，一定要创建 SPFILE。

```
    SQL>  CREATE  SPFILE='C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'     FROM     PFILE='C:\app\Administrator\product\11.1.0\db_3\
database\INITZDB.ORA'
      2  /
    File Created.
```

其中，SPFILEZDB.ORA 就是 SPFILE。INITZDB.ORA 就是 PFILE。

STEP 04 重新启动 Windows XP。

STEP 05 测试数据库是否能够自动启动。

操作系统启动以后，执行下面的过程。

ⓐ 启动 SQL*Plus，登录数据库。

```
C:\Documents and Settings\Administrator> sqlplus
```

ⓑ 访问任意一个表。

```
SQL> SELECT * FROM A;
         A
----------
         1
```

其中，A 是以前存在的一个表。

从上面的过程可以看出，表 A 可以访问，说明数据库自动启动成功。



6.1.10 把数据库设置成自动启动/关闭之——UNIX/Linux 篇

在操作系统（OS）启动时，管理员可以把 Oracle 设置成随操作系统一起自动启动/关闭。Oracle 自带了两个 Shell 程序：dbstart 与 dbshut。dbstart 用于启动数据库；dbshut 用于关闭数据库。dbstart 与 dbshut 会读取/var/opt/oracle/oratab（Solaris）文件，或者/etc/oratab（Linux）文件。这两个 Shell 程序存放在$ORACLE_HOME/bin 目录下。设置数据库自动启动/关闭的过程说明如下。

STEP 01 以用户 oracle 的身份登录操作系统。

```
su – oracle
```

STEP 02 进入启动/关闭脚本所在的目录。

```
$ cd $ORACLE_HOME/bin
```

STEP 03 查看启动脚本（dbstart）/关闭脚本（dbshut）是否存在。

```
$ ls dbstart
dbstart
$ ls dbshut
dbshut
```

STEP 04 查看文件 oratab 的内容。

文件 oratab 存放数据库的信息，包括实例名、Oracle 主目录等信息。

文件 oratab 中将会有类似下面的条目，条目的格式如下所示。

SID:ORACLE_HOME:N(Y)

文件 oratab 放在下面的目录中。

- /var/opt/oracle/oratab（**Solaris**）
- /etc/oratab（**其他的 UNIX、Linux 系统**）

查看 oratab 文件的内容：

```
$ more /var/opt/oracle/oratab
……
*:/oracle/app/oracle/product/11.1.0/Db_1:N
zdb:/oracle/app/oracle/product/11.1.0/Db_1:N
```

……

其中，zdb 是我们要设置成自动启动/关闭的实例的 SID；/oracle/app/oracle/product/11.1.0/Db_1 是 Oracle 的主目录；Y/N 表示是否允许自动启动/关闭实例；Y 表示允许数据库自动启动/关闭。



修改文件 oratab 的内容，使数据库自动启动（关闭）。

```
vi /var/opt/oracle/oratab（Solaris）
vi /etc/oratab（其他 UNIX、Linux 系统）
```

将

```
zdb:/oracle/app/oracle/product/11.1.0/Db_1:N
```

改成

```
zdb:/oracle/app/oracle/product/11.1.0/Db_1:Y
```



验证使用 dbstart 脚本是否可以手工启动数据库。如果这里都不能成功启动数据库，那么操作系统启动时，数据库肯定不能自动启动。

```
$ ./dbstart
SQL*Plus: Release 11.1.0.6.0 - Production on 星期五 7 月 24 17:43:52 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
SQL> Connected to an idle instance.
SQL>
 ORACLE instance started.
Total System Global Area  184549376 bytes
Fixed Size                 778292 bytes
Variable Size            70262732 bytes
Database Buffers        113246208 bytes
Redo Buffers              262144 bytes
Database Mounted.
Database opened.
SQL> Disconnected FROM Oracle Database 10g Enterprise Edition Release
11.1.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options
Database "zdb" warm started.
$
```



得到 Oracle 的主目录（Oracle Home），这个值在随后的步骤中将会用到。

```
$ env
……
ORACLE_HOME=/oracle/app/oracle/product/11.1.0/Db_1
……
```

本例中，Oracle 的主目录是/oracle/app/oracle/product/11.1.0/Db_1。



以用户 root 的身份登录操作系统（OS）。

```
su -
```

STEP 09 创建一个新的脚本 start_shut。

ⓐ 进入下面的目录，将在这个目录下创建一个 Shell 脚本。

```
# cd /etc/init.d
```



注：有的系统是/etc/rc.d/init.d。

ⓑ 创建脚本 start_shut（需要 root 权限）。

```
# vi start_shut
```

在这个文件（参考光盘中的\script\start_shut）中加入如下的内容：

```
#!/bin/sh
    OPT_=$1
    case "$OPT_" in
    start)
        /bin/echo "$0 : (start)"
        #
        # Your service startup command goes here.
        #
        su - oracle -c "/oracle/app/oracle/product/11.1.0/Db_1/bin/lsnrctl
start"
        su - oracle -c "/oracle/app/oracle/product/11.1.0/Db_1/bin/
dbstart"
        # NOTE: Must exit with zero unless error is severe.
        chmod 1777 /tmp
        chown sys:sys /tmp
        exit 0
        ;;
    stop)
        /bin/echo "$0 : (stop)"
        #
        # Your service shutdown command goes here.
        #
        su - oracle -c "/oracle/app/oracle/product/11.1.0/Db_1/bin/lsnrctl
stop"
        su - oracle -c "/oracle/app/oracle/product/11.1.0/Db_1/bin/dbshut"
            # NOTE: Must exit with zero unless error is severe.
        exit 0
        ;;
    *) /bin/echo ''
        /bin/echo "Usage: $0 [start|stop]"
        /bin/echo "      Invalid argument ==> \"${OPT_}\""
        /bin/echo ''
        exit 0
        ;;
    esac
```

其中，/oracle/app/oracle/product/11.1.0/Db_1 表示 Oracle 的主目录（Oracle Home）。如果你 Oracle 的主目录不是/oracle/app/oracle/product/11.1.0/Db_1，则需要对其进行修改。

STEP 10 修改 Shell 脚本 start_shut 的权限，使它可以被执行。

```
# chmod 775 start_shut
# ls -l start_shut
-rwxrwxr-x   1 root     other      1077 Aug  8 11:15 start_shut
```

STEP 11 设置开机时自动启动 Oracle。

ⓐ 进入下面的目录。

```
cd /etc/rc2.d（如果默认的操作系统启动级别是 2）
```

ⓑ 创建连接。

```
ln -s /etc/init.d/start_shut S99start_shut（Solaris）
ln -s /etc/rc.d/init.d /start_shut S99start_shut（其他 UNIX 系统）
```

STEP 12 设置关机时自动关闭 Oracle。

ⓐ 进入下面的目录。

```
#cd /etc/rc0.d
```

ⓑ 创建连接。

```
#ln -s /etc/init.d/start_shut K01start_shut（Solaris）
#ln -s /etc/rc.d/init.d /start_shut K01start_shut(其他 UNIX)
```

STEP 13 重新启动系统，测试 Oracle 是否能够自动启动。

```
# init 6
```

在操作系统（OS）重启的过程中，我们将会看到 Oracle 的启动过程，这个过程和常规的启动过程没有什么区别。启动的时候，也会显示类似下面的信息：

```
ORACLE instance started.
Total System Global Area  289406976 bytes
Fixed Size                 1248600 bytes
Variable Size            104858280 bytes
Database Buffers         180355072 bytes
……
```


6.2 数据库启动故障排查攻略

我们都知道，Oracle 数据库以其功能强大和稳定而驰骋于业内。尽管如此，我们在使用数据库的过程中，由于系统本身的问题，或由于人为操作，会导致各种各样的数据库故障（系统级的故障，包括数据启动时的故障、数据库关闭时的故障等）。如何诊断这些故障，下面我们将给出一般规律（即为这类故障的排查方法）。

如图 6-5 所示，数据启动故障诊断由两个阶段组成：**检查数据库**、**检查操作系统**（主要检查操作系统的日志文件）。检查数据库包括检查**警报文件**（Alert Log）和**跟踪文件**（Trace

Log）。

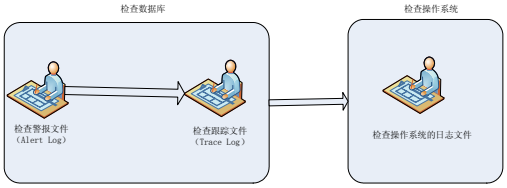


图 6-5　数据库启动故障诊断

知识点索引——警报文件（Alert Log）

警报日志文件，本书简称警报文件，它用于记录数据库的重大活动和发生的错误。警报文件按照时间的先后记录发生的事件。特别注意的是，警报文件除了记录数据库中发生的错误外，还记录数据库中发生的重大事件。警报文件记录的内容如下：

- ✓ 每次数据库启动（STARTUP）和关闭（SHUTDOWN）的详细信息。
- ✓ DBA 执行的某些管理操作，如 ALTER SYSTEM，ALTER DATABASE。
- ✓ 某些数据库错误，如 Oracle 的内部错误（ORA-600）、空间错误等。
- ✓ 共享服务器相关的信息和错误。
- ✓ 值是非默认值的初始化参数（Initialization Parameters）信息。
- ✓ 物化视图（Materialized View）自动刷新产生的错误。

警报文件的名字的格式是 alertSID.log，其中，SID 表示实例名（Instance）。警报文件的位置由初始化参数 BACKGROUND_DUMP_DEST 指定。

通过分析警报文件的内容，我们可以了解数据库发生的重大事件和错误。警报文件是 DBA 使用最频繁的文件，DBA 常常参考该文件进行故障排查。

知识点索引——跟踪文件（Trace Log）

跟踪日志文件，本书简称跟踪文件，每个服务器进程和后台进程都写跟踪文件。当一个后台进程检测到错误的时候，Oracle 会把错误信息写到跟踪文件中。因此，检查跟踪文件的内容，就可以知道后台进程运行中产生的错误。

跟踪信息被写到两个目录中，和后台进程（Background Processes）相关的信息被写到初始化参数 DIAGNOSTIC_DEST 指定目录；和服务器进程（Server Processes）相关的信息被写到初始化参数 USER_DUMP_DEST 指定的目录。跟踪文件的名字在每种操作系统上会有所不同，但是，每个跟踪文件的名字都包含进程的名字（如 LGWR）。

如果数据库长时间运行，跟踪文件会变得越来越大。我们可以手工清除跟踪文件，

也可以限制跟踪文件的大小。Oracle 的初始化参数 MAX_DUMP_FILE_SIZE 用于限制跟踪文件的大小。MAX_DUMP_FILE_SIZE 的格式如下：

```
MAX_DUMP_FILE_SIZE = { integer [K | M] | UNLIMITED
```

其中 K｜M 表示单位，可以把跟踪文件的大小设置在多少 KB 或多少 MB，也可以设置跟踪文件无限制（UNLIMITED）增长。



知识点索引——查看操作系统的日志文件

每个操作系统都有自己的日志文件，这些操作系统文件记录了操作系统的系统级改变和错误信息，从这些文件中，我们能够获得一些提示，这些提示能够帮助我们分析和定位数据库安装和运行中出现的问题。

操作系统日志文件的读取方式

- ✓ 大部分日志文件可以直接读取（使用命令 more、cat 等）
- ✓ 有的操作系统日志文件要使用特殊的命令才能读取

用 more 或者其他文本查看命令来查看 errlog 文件（AIX 日志文件），我们看到的将是一堆乱码。

操作系统日志文件的位置

在 Linux 中，**/etc/syslog.conf** 记录了日志文件的位置。大部分日志文件都位于目录/var/log/之下。Linux 中存在如下日志文件。

- ✓ /var/log/messages　将 info 或更高级别的消息写到该文件
- ✓ /var/log/maillog　记录电子邮件相关的信息
- ✓ /var/log/boot.log　记录系统启动相关的信息
- ✓ /var/log/cron　记录系统中定时执行的任务相关的信息

还有其他日志文件，这里就不一一介绍。

Solaris 的日志文件通常位于目录/var/adm 下，Solaris 有如下的日志文件：

- ✓ messags 记录来自系统核心的各种运行日志
- ✓ syslog　记录 mail 事件
- ✓ sulog　记录普通用户使用 su 命令切换到其他用户的记录
- ✓ utmp　记录当前登录在主机上的用户

通常，我们首先查看 Oracle 警报文件和跟踪文件是否有异常。如果信息还不足以诊断错误，我们可以查看操作系统的日志文件，然后再结合当前数据库的各种状态信息，相信我们一定能解决数据库的故障。



注：这种常规的诊断方法适合诊断数据库启动、关闭及其他数据库级别的故障。

6.3 数据库的关闭

数据库关闭有四种方式：NORMAL、IMMEDIATE、TRANSACTIONAL、ABORT。四种方式各有千秋，以下将逐一介绍。

6.3.1 使用频率最高的数据库关闭方式

IMMEDIATE 方式是数据库关闭使用频率最高的方式。以 IMMEDIATE 方式关闭数据库时，会发生下面的事情：

- 新的用户不能注册（登录）数据库。
- 未提交的事务将被回滚（Rolled Back）。
- Oracle 不会等待所有的用户（连接）退出数据库。

以 IMMEDIATE 方式关闭数据库具有以下特点：

- 以 IMMEDIATE 方式关闭数据库不需要实例恢复（Instance Recovery）。
- SHUTDOWN IMMEDIATE 是最安全的数据库关闭方式。
- 使用 SHUTDOWN IMMEDIATE 关闭数据库的过程比较慢。

使用 IMMEDIATE 方式关闭数据库的过程说明如下。

STEP 01 启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```

STEP 02 以 IMMEDIATE 方式关闭数据库。

```
SQL> SHUTDOWN IMMEDIATE;
```

知识点索引——数据库关闭原理

数据库启动要经历三个阶段，数据库关闭也要经历三个阶段。

阶段一：关闭数据库（Close The Database）；

阶段二：卸载数据库（Unmount The Database）；

阶段三：关闭实例（Shut Down The Instance）。

✓ **关闭数据库**

关闭数据库的时候，Oracle 首先把 SGA 中的数据写到数据文件和重做日志文件中。然后，Oracle 关闭所有的数据文件和重做日志文件。这时候，数据库已经不可访问。

这个阶段完成以后，控制文件仍然处在打开状态。

✓ **卸载数据库**

数据库关闭完成以后，Oracle 将分离数据库和实例之间的联系，这个阶段叫“卸载

数据库”或者叫“UNMOUNT 数据库”。

这个阶段仅仅是卸载数据库，实例仍然存活在内存中。

这个阶段完成后，控制文件被关闭。

✓ **关闭实例**

这是关闭数据库的最后一个阶段，这个阶段 Oracle 将从内存中移出 SGA 并终止正在运行的后台进程（Background Processes）。至此，数据库关闭已经完成。

6.3.2 数据库关闭最快的方式

ABORT 方式是关闭数据库最快的方式。以 ABORT 方式关闭数据库时，会发生下面的事情：

- 不允许启动新的连接（New Connections）和新的事务（New Transactions）。
- 客户端的 SQL 语句（Client SQL Statements）立刻中止。
- 未提交的事务不被回滚（RollBack）。
- Oracle 立刻中止所有连接（会话）。

以 ABORT 方式关闭数据库具有以下特点：

- 只有数据库出现问题的时候，才使用这种方式关闭数据库。
- 这是一种最不安全的关闭方式，数据库重启时需要实例恢复（Oracle 后台进行）。
- 以 ABORT 方式关闭数据库是最快的一种方式。

以 ABORT 方式关闭数据库的过程说明如下。

STEP 01 启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```

STEP 02 以 ABORT 方式关闭数据库。

```
SQL> SHUTDOWN ABORT;
```

注：如果使用 ABORT 方式关闭数据库，下次数据库启动时，需要实例恢复（Instance Recovery）。因此，不推荐使用这种方式关闭数据库。

6.3.3 数据库关闭最慢的方式

以 NORMAL 方式关闭数据库不是很常用。因为，如果使用 NORMAL 方式关闭数据库，也许有的数据库永远也不能关闭。以 NORMAL 关闭数据库时，不需要实例恢复（Instance Recovery）。SHUTDOWN NORMAL 是最慢的一种数据库关闭方式。

使用 NORMAL 方式关闭数据库时，有以下特点：

- 允许新的用户注册（登录）数据库。
- 要等所有的用户自动退出 Oracle 以后，Oracle 才关闭数据库。所以，如果数据库中

存在一个用户，那么，Oracle 就一直等待，直到该用户退出，Oracle 才关闭数据库。使用 NORMAL 方式关闭数据库的过程说明如下。

STEP 01 启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```

STEP 02 以 NORMAL 方式关闭数据库。

```
SQL>SHUTDOWN NORMAL(或者 SHUTDOWN)
```



注：使用 NORMAL 方式常常不能关闭数据库 ,不推荐用 NORMAL 方式关闭数据库。

6.3.4 以 TRANSACTIONAL 方式关闭数据库

以 TRANSACTIONAL 方式关闭数据库时，会发生下面的事情：

- 不允许新的用户注册（登录）数据库；
- 不允许建立新的事务（New Transactions）；
- 所有的事务（Transactions）完成以后才关闭数据库；
- 一个用户（会话）执行完手里的事务（Transactions）后将被强行断开与数据库的联机。

以这种方式关闭数据库有以下特点：

- 这种关闭方式不会使客户端的数据丢失；
- 这种关闭方式不需要实例恢复（Instance Recovery）；
- 使用 SHUTDOWN TRANSACTIONAL 命令关闭数据库的过程比较慢。

使用 TRANSACTIONAL 方式关闭数据库的过程说明如下。

STEP 01 启动 SQL*Plus，以 SYSDBA 的身份登录数据库。

```
SQL> CONNECT sys AS SYSDBA
```

STEP 02 以 TRANSACTIONAL 方式关闭数据库。

```
SQL> SHUTDOWN TRANSACTIONAL;
```




本章总结

本章详细介绍了数据库启动和关闭的原理，并教会大家如何把数据库设置成自动启动和关闭。如果大家有时间，可以好好理解数据库的启动和关闭原理；如果仅仅停留在数据库的操作上，那么只需要知道如何启动和关闭数据库即可。启动数据库时，我们会用到 PFILE 和 SPFILE，我们应该学会使用 PFILE 和 SPFILE 来控制数据库。

第 7 章　数据之家——Oracle 存储管理

本章重点介绍表空间和数据文件，并理解 Oracle 存储的物理结构和逻辑结构。

7.1　常规表空间管理

本节将介绍常规表空间的使用，如如何创建、扩展和删除表空间等。

7.1.1　创建表空间（Create Tablespace　）

在讲述表空间之前，让我们先感受一下创建表空间的过程。创建表空间的过程如下。

STEP 01　确保数据库控件控制台进程（Database Control Console Process）dbconsole 正在运行。

有关进程 dbconsole 的信息，**请参考本书 5.2.1 节的内容。**

STEP 02　打开浏览器，从企业管理器登录远程数据库。

在浏览器中输入网址（本例中是 https://zero:1158/em），输入用户名和密码（本例中的用户名和密码是 system/zero），登录数据库。

登录数据库后，单击“服务器”→“表空间”，并单击 创建 按钮，出现如图 7-1 所示的界面。



图 7-1　设置表空间参数

STEP 03　设置表空间的参数。

输入表空间名字（userdb），单击 添加 按钮，出现如图 7-2 所示的界面。

STEP 04　添加数据文件。



图 7-2　添加数据文件

在“文件名”处输入数据文件的名字（userdata1），在“文件目录”处输入数据文件的路径（F:\NEWDB\ZDB\），选中“数据文件满后自动扩展（AUTOEXTEND）”。在“增量”处输入 20，选择单位是 MB。

单击 继续 按钮，再单击 确定 按钮，开始创建表空间。创建完成后，出现如图 7-3 所示的界面。

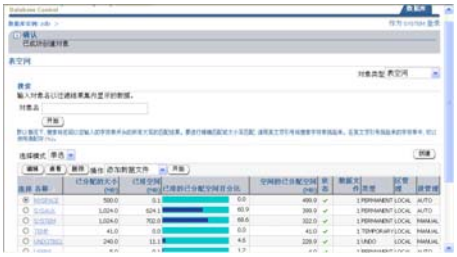


图 7-3　表空间创建完成

STEP 05 创建完成。

从左上角的提示信息“已成功创建对象”可以看出，表空间创建成功。当然，我们也可以使用下面的脚本手工创建表空间 userdb。

```
CREATE TABLESPACE userdb
DATAFILE 'F:\NEWDB\ZDB\userdata1' SIZE 200M AUTOEXTEND ON NEXT 20M
MAXSIZE UNLIMITED LOGGING EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO DEFAULT
```

- CREATE TABLESPACE 是关键字，表示创建表空间。这里创建了一个名为 userdb 的表空间。
- DATAFILE 是关键字，用于指定组成表空间的数据文件，表空间由一个数据文件 userdata1 组成。数据文件的大小（SIZE）是 200MB，数据文件被设置成以 20MB 为单位自动扩展（AUTOEXTEND），该文件可以无限制（UNLIMITED）扩展。
- EXTENT MANAGEMENT LOCAL 表示这是一个本地管理的表空间。
- SEGMENT SPACE MANAGEMENT AUTO 使用段的自动管理。

从上面还可以看出，上面的表空间由一个数据文件组成。通常，表空间由一个或者多个数据文件组成，表空间和数据文件的关系如图 7-4 所示。

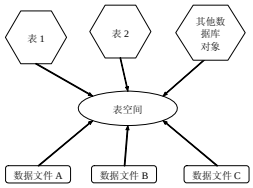


图 7-4　表空间和数据文件的关系

表空间是 Oracle 存储的逻辑结构，数据文件是 Oracle 存储的物理结构。

知识点索引——表空间的定义

一个数据库被分成一个个的逻辑单元，这些逻辑单元叫“表空间”。表空间用于存放数据库对象（如表、索引等）。一个数据库逻辑上由一个或者多个表空间组成。表空间由一个或者多个数据文件（物理结构）组成（如图 7-4 所示）。表空间用于存放数据库对象（如图 7-4 所示），而这些对象实际上存放在数据文件中，数据文件是数据的物理载体。表空间的尺寸是组成这个表空间的所有数据文件的尺寸之和。

每个数据库都拥有一个系统表空间（SYSTEM Tablespace）和一个辅助表空间（SYSAUX Tablespace）。创建数据库的时候，Oracle 会自动创建这两个表空间。在默认情况下，Oracle 创建小文件表空间（Smallfile Tablespace），小文件表空间是传统的表空间。系统表空间和辅助表空间是小文件表空间。同时，Oracle 还允许创建大文件表空间（Bigfile Tablespace）。大文件表空间是由单个很大的文件组成的表空间。

数据库中有以下类型的表空间：永久表空间（Permanent Tablespaces，又叫常规表空间）、临时表空间（Temporary Tablespace）、回滚表空间（Undo Tablespace）。本章对它们进行一一介绍。



知识点索引——数据文件的定义

一个数据库有多个数据文件。每个数据文件都是一个操作系统文件。数据文件真正存放数据库的数据。数据库中的表（Tables）索引（Indexes）的数据，物理上存放在数据文件中。

一个数据文件由多个操作系统块（OS Block）组成，如图 7-5 所示。

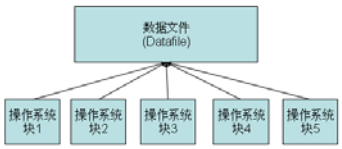


图 7-5　数据文件的组成



注：操作系统块和 Oracle 的数据块是两个概念。

数据文件有下面的特征：

- ✓ 一个数据文件只能属于一个数据库。
- ✓ 数据文件可以被设置成自动扩展。如：10MB 的数据文件，当该文件上的空间使用完以后，它可以根据需要自动扩展。
- ✓ 一个或者多个数据文件形成一个表空间（表空间是 Oracle 的逻辑结构）。
- ✓ 一个数据文件只能属于一个表空间。

用户查询一个表，如果这个表的数据不在内存中，Oracle 会读取该表所在的数据文件，并把数据放到内存中。



知识点索引——表空间中区（Extent）的管理方式

对于表空间中区的空间管理，Oracle 有两种管理方式：

- ✓ 字典管理（Dictionary Managed Tablespace），通过数据字典对区进行管理。
- ✓ 本地管理（Locally Managed Tablespace），通过位图对区进行管理。

本地管理是 Oracle 推荐使用的区的管理方式。

■ 字典管理（Dictionary Managed Tablespaces）

如果使用字典管理，Oracle 将使用数据字典来跟踪表空间中区的使用。当分配一个新的区、使用一个区或者释放一个区的时候，Oracle 都会更新数据字典 FET$与 UET$。通过数据字典来反映区的状态。当更新数据字典时，会产生回滚信息，Oracle 会存储这些回滚信息。这正是字典管理的表空间逐渐被淘汰的根源。

■ 本地管理（Locally Managed Tablespaces）

本地管理表空间是 Oracle 推荐使用的方式，它通过本地表空间来管理区的使用。

表空间中的每个数据文件都有一个位图（Bitmap），位图存放在数据文件头，位图中的每个位（bit）与一个或者一组数据块（Block）相对应。当分配一个区或者释放一个区的时候，Oracle 将更新位图以反映这种改变。特别注意的是，对于本地管理的表空间，Oracle 不会更新数据字典，也不会产生回滚信息。



知识点索引——段（Segment）的管理方式

当用户进行数据的插入、更新、删除的时候，段中数据块的空间会发生改变，块的状态也会发生改变。Oracle 使用两种方式来管理这些块：

- ✓ 段的手动管理（Manual Segment Space Management）；
- ✓ 段的自动管理（Automatic Segment Space Management，ASSM）。

■ 段的手动管理

Oracle 使用自由列表（Free List）来管理段中的块。自由列表列出了允许进行插入操作的数据块。

Oracle 通过 PCT_FREE（PCTFREE）和 PCT_USED（PCTUSED）两个参数来控制一个数据块是否允许插入数据。当对一个块进行 INSERT 或者 UPDATE 操作以后，Oracle 会把这个块中剩余的自由空间和 PCT_FREE 进行对比，如果这个数据块中的自由空间小于 PCT_FREE 的设置，则 Oracle 将把这个块从自由列表中拿出来，表示这个块不再允许进行插入，但是，这个块中剩余空间允许更新操作；当对一个块进行 DELETE 或者 UPDATE 操作以后，Oracle 会把这个块中已使用的空间和 PCT_USED 进行对比，如果这个数据块中已经使用的空间小于 PCT_USED 的设置，则 Oracle 把这个块放回自由列表中，表示这个块允许数据的插入。因此，PCTFREE 涉及把块从自由列表中移出，PCTUSED 涉及把块移到自由列表中。

通过 SEGMENT SPACE MANAGEMENT MANUAL 可以设置段的手动管理，例如：

```
CREATE TABLESPACE tsseg_manual
DATAFILE 'D:\mydata\tsseg_manual.DBF' SIZE 10G
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT MANUAL;
```

MANUAL 指定在表空间 tsseg_manual 中段的管理方式是手动管理。

■ 段的自动管理

“段的手动管理”使用自由列表来管理段中数据块的使用，与之相比，“段的自动管理”使用位图来管理段中的数据块。当块中的空间发生变化的时候，Oracle 会更新位图，以反映这块是否允许数据的插入。位图使 Oracle 能够自动管理自由空间。

创建表空间时，可以使用 SEGMENT SPACE MANAGEMENT AUTO 来把段设置成自动管理。

7.1.2 扩展表空间

在系统运行一段时间后，表空间已经耗尽，只有扩展表空间，才能继续在表空间上添加新表，才能往已经存在的表中添加数据。表空间扩展的实质是扩展组成表空间的数据文件（数据文件是实际存在于物理磁盘上的文件）。有三项措施可以用于表空间的扩展。

■ 措施 1：手动增加表空间中数据文件的大小。

例如：

```
ALTER DATABASE DATAFILE ' d:\new\product1' RESIZE 1000M
```

把表空间中数据文件的尺寸扩大到 1000MB。

其中，d:\new\ product1 是组成表空间的数据文件。

下次表空间用完的时候，我们还得手动增加表空间的空间。

■ 措施 2：把表空间中的数据文件设成自动扩展。

例如：

```
ALTER DATABASE DATAFILE ' d:\new\product1' AUTOEXTEND ON NEXT 5M MAXSIZE
UNLIMITED
```

其中，d:\new\product1 是组成表空间的数据文件；AUTOEXTEND ON 表示这个数据文件可以自动扩展，“自动扩展”的意思是当这个数据文件（product1）的空间用完时，这个数据文件的空间可以自动增加; NEXT 5M 指定这个数据文件（product1）每次扩展的大小（增量）；MAXSIZE UNLIMITED 表示这个数据文件（product1）的空间可以无限制扩展。

■ 措施 3：往表空间中增加数据文件。

例如：

```
ALTER TABLESPACE userdb ADD DATAFILE 'd:\new\p2.ora' SIZE 100M
```

往表空间 userdb 中增加数据文件 p2.ora，这样同样也可以实现扩展表空间的目的。

7.1.3 使表空间离线/在线（Taking Tablespaces Offline/Online）

除了 SYSTEM 表空间外，在数据库打开的时候，管理员可以让任何表空间离线或者在线。

- 所谓“**表空间在线（Tablespace Online）**”就是使表空间中的数据可以访问（Accessible）。在数据库打开的状态下，SYSTEM 表空间永远在线，因为 Oracle 需要访问存放在 SYSTEM 表空间中的数据字典（Data Dictionary）。
- 所谓“**表空间离线（Tablespace Offline）**”就是使表空间中的数据不可访问（Not Accessible）。管理员使一个表空间离线，通常是出于维护的目的.

■ 使表空间离线

```
ALTER TABLESPACE userdb OFFLINE;
```



知识点索引——表空间离线的 3 种方式

使表空间离线的三种方式：

- ✓ NORMAL 常规的离线方式，会执行校验点（Checkpoint），是比较安全的一种方式（ALTER TABLESPACE userdb OFFLINE NORMAL）。
- ✓ TEMPORARY 数据文件有错误时，应使用这种方式（ALTER TABLESPACE userdb OFFLINE TEMPORARY）。
- ✓ IMMEDIATE 离线数据文件，但是不会产生校验点，表空间重新在线时，需要执行恢复（ALTER TABLESPACE userdb OFFLINE IMMEDIATE）。

■ 使表空间在线

```
ALTER TABLESPACE userdb ONLINE;
```



注：表空间离线具有一定的风险性。



7.1.4 重命名表空间（Renaming Tablespaces）

重命名表空间的例子如下所示：

```
ALTER TABLESPACE userdb RENAME TO userdb2;
```

上面的例子把表空间的名字由 userdb 改成 userdb2。

表空间重命名注意事项：

- ➢ 不能重名表空间 SYSTEM、SYSAUX。
- ➢ 如果表空间离线或者表空间中有数据文件离线，表空间不能被重命名。



7.1.5 使表空间只读（Read Only）

使表空间只读的例子如下所示：

```
ALTER TABLESPACE userdb READ ONLY;
```

上述命令使表空间 userdb 只读。



知识点索引——只读表空间（Read Only Tablespace）

只读表空间，顾名思义，这个表空间只允许读，不允许修改。鉴于只读表空间的这种特性，只读表空间可以放在只读的设备上，如 CD-ROM、WORM。

如果要想修改一个只读表空间，只能先把这个表空间的状态设成可读/写（Read/Write），接着对该表空间上的数据进行修改，修改完成后，再把该表空间改成只读（Read Only）。

由于只读表空间很少被修改，所以只读表空间不需要频繁备份。

如果想去除表空间的只读属性，只需输入如下命令：

```
ALTER TABLESPACE userdb READ WRITE;
```

上面的 SQL 语句使表空间 userdb 可读写。



7.1.6 删除表空间（Drop Tablespace）

删除表空间必须具有 DROP TABLESPACE 权限。

```
DROP TABLESPACE userdb INCLUDING CONTENTS;
```

如果删除空的表空间，只需输入 DROP TABLESPACE userdb 即可；如果表空间中有内容，必须使用选项 INCLUDING CONTENTS。

删除表空间及其数据文件：

```
DROP TABLESPACE userdb INCLUDING CONTENTS AND DATAFILES;
```



7.1.7 大文件表空间（Bigfile Tablespaces）

创建大文件表空间 biguserdb，表空间大小是 50GB。

```
CREATE BIGFILE TABLESPACE biguserdb
DATAFILE ' F:\NEWDB\ZDB\userdata1' SIZE 50G
```

BIGFILE 指定创建的表空间是大文件表空间。



知识点索引——大文件表空间（Bigfile Tablespace）

大文件表空间由一个单独很大的数据文件组成。大文件表空间使 Oracle 能够使用和管理超级大的文件。

在默认情况下，系统默认创建小文件表空间（Smallfile Tablespace），小文件表空间是传统的表空间类型。Oracle 创建的 SYSTEM 表空间和 SYSAUX 表空间都是小文件表空间。

使用大文件表空间有下面的优点。

- ✓ 使表空间的容量更大。
- ✓ 在超大型数据库中，简化数据文件的管理。使用大文件表空间，可以减小 SGA 的需求。同时，还可以减少控制文件的尺寸。
- ✓ 简化数据库的管理。

7.1.8 表空间的加密（Encrypted Tablespaces）

加密表空间能够有效地保护敏感数据。有以下的几种加密算法：3DES168、AES128、AES192、AES256，它们都可以用于加密表空间。

```
CREATE TABLESPACE securespace
DATAFILE 'F:\NEWDB\ZDB\userdata1' SIZE 100M
ENCRYPTION USING 'AES256'
DEFAULT STORAGE(ENCRYPT);
```

创建加密的表空间 securespace，使用加密算法‘AES256’。

7.1.9 表和表空间的关系

如图 7-4 所示，表被存放在表空间，从逻辑结构的角度，表空间是表的容器；从物理结构的角度，表实际上存放在数据文件，数据文件是表的物理载体。当创建表时，我们可以在 CREATE TABLE 语句中指定表存放在指定的表空间。

1．列出数据库中有哪些表空间

数据字典 DBA_TABLESPACES 存放表空间的信息，从该视图中可以知道数据库中有哪些表空间。

```
SQL> select tablespace_name FROM DBA_TABLESPACES;
```

SQL 语句的执行结果如下所示：

```
TABLESPACE_NAME
------------------------------
SYSTEM
SYSAUX
UNDOTBS1
…
```

2．查出组成表空间的数据文件

数据字典 DBA_DATA_FILES 存放数据文件的信息，从该视图中，可以知道一个表空间由哪些数据文件组成。

```
SQL> SELECT FILE_NAME FROM DBA_DATA_FILES
  2  WHERE  TABLESPACE_NAME='MYUNDOTBS';
```

SQL 语句的执行结果是：

```
FILE_NAME
-------------------------------------------
C:\ZHAO\VMDATA\UNDO0201.DBF
```

```
C:\ZHAO\VMDATA\UNDO0202.DBF
SQL>
```

从上面可以看出，表空间 myundotbs 由数据文件 UNDO0201.dbf 和 UNDO0202.dbf 组成。

3．表空间的剩余空间

通过连接（Join）视图 DBA_FREE_SPACE 和 DBA_DATA_FILES，可以查询出一个表空间的使用情况，包括该表空间的总共大小、已使用空间、剩余空间、已使用百分比、剩余百分比（参考光盘中的文件 free_space.sql）。

```
SQL> SELECT tbs  表空间名,
  2  sum(totalM) 总共大小 M,
  3  sum(usedM) 已使用空间 M,
  4  sum(remainedM) 剩余空间 M,
  5  sum(usedM)/sum(totalM)*100 已使用百分比,
  6  sum(remainedM)/sum(totalM)*100 剩余百分比
  7  FROM (
  8     SELECT b.file_id    ID,
  9     b.Tablespace_name    tbs,
 10     b.file_name         name,
 11     b.bytes/1024/1024            totalM,
 12     (b.bytes-sum(nvl(a.bytes,0)))/1024/1024     usedM,
 13     sum(nvl(a.bytes,0))/1024/1024              remainedM,
 14     sum(nvl(a.bytes,0))/(b.bytes)*100  剩余百分比 ,
 15     (100 - (sum(nvl(a.bytes,0))/(b.bytes)*100))  已使用百分比
 16     FROM dba_free_space a,dba_data_files b
 17     WHERE a.file_id=b.file_id
 18     GROUP BY b.Tablespace_name,b.file_name,b.file_id,b.bytes
 19     ORDER BY b.Tablespace_name
 20  )
 21  GROUP BY tbs
 22  /
```

SQL 语句的执行结果如下所示：

```
表空间名       总共大小 M   已使用空间 M    剩余空间 M  已使用百分比   剩余百分比
----------   ---------- ----------------------------------------
UNDOTBS1      555       20.3125     534.6875    3.65990991  96.3400901

SYSAUX        607.375    576.8125    30.5625     94.9681004  5.03189957

USERS         5          .0625       4.9375      1.25        98.75

表空间名       总共大小 M 已使用空间 M     剩余空间 M     已使用百分比    剩余百分比
----------
SYSTEM        690        687.625      2.375       99.6557971  .344202899
MYUNDOTBS      5         1.0625      3.9375      21.25       78.75
SQL>
```

从上面的输出可以看出，表空间 SYSAUX 总共大小是 607.375MB，已使用 576.8125MB，剩余 30.5625MB，已使用百分比是 94.9681004%，剩余百分比是 5.03189957%。

4．得到创建表空间的 SQL 语句

如果想重建一个表空间，我们可以先获得创建该表空间的 DDL 语句，然后利用得到的 DDL 语句重建表空间。通过包 DBMS_METADATA 的过程 GET_DDL 可以得到创建表空间的 DDL 语句。

STEP 01 以用户 system 的身份登录数据库。

```
SQL> CONNECT system
Enter password:
Connected.
```

STEP 02 设置 SQL*Plus 的环境变量。

```
SQL> set long 10000
```

如果不进行设置，执行 DBMS_METADATA.GET_DDL 将会产生错误。

STEP 03 得到创建表空间的 SQL 语句。

```
SQL> SELECT DBMS_METADATA.GET_DDL('TABLESPACE','MYSPACE') FROM dual;
DBMS_METADATA.GET_DDL('TABLESPACE','MYSPACE')
--------------------------------------------------------------------------------
  CREATE TABLESPACE "MYSPACE" DATAFILE
  'D:\ZHAO\VMDATA\MYSPACE2' SIZE 524288
```

得到创建表空间 MYSPACE 的 SQL 语句。

7.1.10 必须清楚的几个概念

要想理解表空间，必须理解 Oracle 的存储结构。Oracle 的存储结构分为物理结构（Physical Structure）和逻辑结构（Logical Structure）。

1．物理结构

一个数据库的物理结构由数据文件（Datafiles）、重做日志文件（Online Redo Logs）和控制文件（Control Files）组成，如图 7-6 所示。

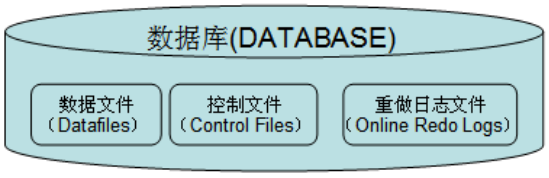


图 7-6 数据库的物理结构

上面是数据库严格意义上的“物理结构组成”，除此之外，组成 Oracle 物理结构的还有一些其他额外的文件，这些文件包括归档日志文件（Archive Log Files）、参数文件（Parameter Files）、警报文件（Alert Files）和跟踪文件（Trace Files）、备份文件（Backup Files）。这些文件将在相关章节进行介绍。

2．逻辑结构

Oracle 存储的逻辑结构由数据块（Data Block）、区（Extent）、段（Segment）和表空间（Tablespace）四部分组成，如图 7-7 所示。

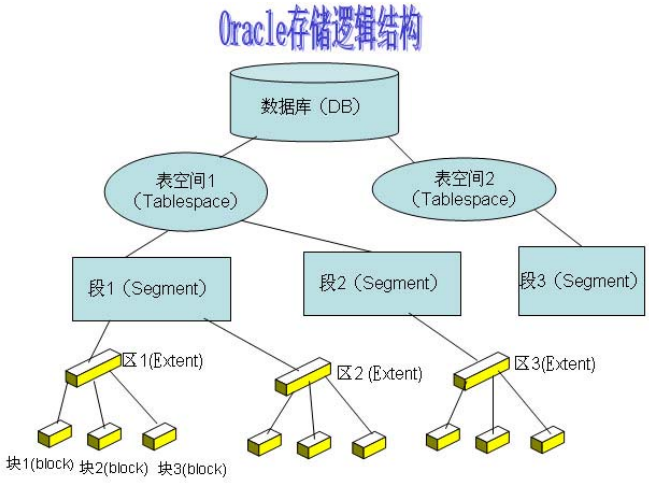


图 7-7　Oracle 存储逻辑结构

块是最基本的存储单元，块组成区，区组成段，段组成表空间，表空间组成数据库。

■ 数据块（Data Block）

数据块，本书简称为“块”，它是 Oracle 最小的存储单位。

■ 区（Extent）

区（Extent）是数据库的一个逻辑存储单元，它由连续的数据块组成。块（Block）组成区，区组成段（Segment）。特别强调的是，区是一段连续的存储空间。当段中的空间耗尽时，Oracle 会分配一个新的区给这个段。有关块、区、段三者之间的关系，请参考图 7-7。

■ 段（Segment）

一个段由一些系列的区组成，段是存放数据的逻辑存储单元。段有几种类型，有数据段（Data Segment）、索引段（Index Segment）、临时段（Temporary Segment）、回滚段（Rollback Segment）。

表由段组成，一个表由一个或者多个段组成。普通表由一个段组成，分区表由多个段组成。表和段的关系如图 7-8 所示。

1．数据段

数据段用于存放数据。一个单独的段可以存放下面的数据：

- 一个普通表的数据
- 分区表（Partition Table）中的一个分区
- 聚簇表（Cluster Table）

创建表的时候，其实质就是创建一个或者多个段。创建数据段的时候，可以指定数据段的存储参数。

2．索引段

索引段用于存放索引数据，每个非分区索引（Nonpartitioned Index）有一个单独的索引段。可以使用 CREATE INDEX 命令创建索引，实质上是创建一个或者多个索引段。创建索引段的时候，可以指定索引段的存储参数。

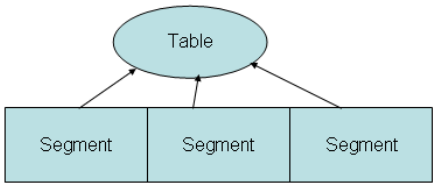


图 7-8　表和段的关系

3．临时段

临时段用于存放临时数据。当进行查询的时候，Oracle 会用到临时段。临时段是 Oracle 自动创建和维护的。临时段用于数据的排序（Sort）。只有内存不足的时候，Oracle 才会用到临时段。Oracle 用到临时段，是数据库性能降低的一个征兆。所以，DBA 应该想方设法尽量让排序在内存中进行。

在使用 CREATE USER 命令创建用户或者使用 ALTER USER 命令修改用户的属性时，可以利用子句 TEMPORARY TABLESPACE 为用户指定临时表空间。如果没有为用户指定临时表空间，Oracle 会默认使用 SYSTEM 表空间（系统表空间）作为用户的临时表空间。SYSTEM 表空间主要用于存放数据字典，鉴于此，我们推荐每个用户使用非系统表空间作为临时表空间。所有分配的临时段属于用户 SYS，不属于执行操作的用户。

4．回滚段

回滚段用于保存回滚条目，Oracle 将数据被修改以前的版本保存在回滚条目中，利用这些条目，用户可以撤销以前对数据的修改。回滚条目还可以实现读一致性。

7.2　数据文件管理

前面一节主要讲述表空间的管理，本节将讲述数据文件的管理。

7.2.1　使数据文件离线

数据文件离线（Datafile Offline）的原理和表空间离线的原理是相同的。表空间离线时，表空间中所有的数据文件都离线。我们也可以使表空间中的单个数据文件离线。离线的数据文件不可访问。在非归档模式下，不要轻易离线数据文件，在这种模式下离线数据文件是很危险的，这样离线的数据文件可能永远也不能重新在线（Online）。在线（Online）

数据文件使该数据文件中的数据可以访问。

➢ 在归档模式下（ARCHIVELOG）使数据文件离线（Offline）

```
ALTER DATABASE DATAFILE 'F:\NEWDB\ZDB\userdata1' OFFLINE;
```

本例中，使数据文件 userdata1 离线。

➢ 在非归档模式下（NOARCHIVELOG）使数据文件离线（Offline）

```
ALTER DATABASE DATAFILE ' F:\NEWDB\ZDB\userdata1' OFFLINE FOR DROP;
```

本例中，使数据文件 userdata1 离线。

如果使用 FOR DROP 选项，这样离线的数据库文件将永远不能再次在线。所以在非归档模式下离线数据文件要非常慎重！

➢ 使数据文件在线（ONLINE）

数据文件离线后，可以使用下面的命令使数据文件重新在线。

```
ALTER DATABASE DATAFILE ' F:\NEWDB\ZDB\userdata1' ONLINE;
```

本例使数据文件 userdata1 重新在线。



7.2.2 移动数据文件

在 Oracle 中，允许把一个数据文件从一个物理位置移动到另外一个物理位置，也允许把一个数据文件从一个目录移动到另外一个目录。移动的方法说明如下（本例将把数据文件 PRODUCT2 从 F:\NEWDB\DATDAT\WM\移动到 F:\NEWDB\SECOND\）。这种方法也可用于重命名数据文件。

在数据库打开的情况下，执行下面的步骤。

查询数据文件的当前位置。

```
SQL> SELECT file_name,Tablespace_name FROM DBA_DATA_FILES;
```

SQL 语句的执行结果如下所示：

```
FILE_NAME                                    TABLESPACE_NAME
---------------------------------------      --------------------
F:\NEWDB\DATDAT\WM\USERS01.DBF                   USERS
F:\NEWDB\DATDAT\WM\SYSAUX01.DBF              SYSAUX
F:\NEWDB\DATDAT\WM\UNDOTBS01.DBF                 UNDOTBS1
F:\NEWDB\DATDAT\WM\SYSTEM01.DBF                  SYSTEM
F:\NEWDB\DATDAT\WM\PRODUCT1                      PRODUCT
F:\NEWDB\DATDAT\WM\TBSK1.ORA                     TBSK2
F:\NEWDB\DATDAT\WM\TBSK2.ORA                     TBSK2
F:\NEWDB\DATDAT\WM\PRODUCT2                      PRODUCT
8 rows Selected
```

数据文件 PRODUCT2 的当前位置是 F:\NEWDB\DATDAT\WM\，数据文件 PRODUCT2 属于表空间 product。

STEP 02 使数据文件 PRODUCT2 所属的表空间 product 离线。

```
SQL> ALTER TABLESPACE product OFFLINE NORMAL;
Tablespace Altered
```

STEP 03 使用操作系统命令把数据文件 PRODUCT2 从 F:\NEWDB\DATDAT\WM\移动到 F:\NEWDB\SECOND。

STEP 04 修改数据文件 PRODUCT2 在控制文件中的信息。

```
SQL> ALTER TABLESPACE PRODUCT
  2  RENAME DATAFILE 'F:\NEWDB\DATDAT\WM\PRODUCT2'
  3  TO 'F:\NEWDB\SECOND\PRODUCT2';
Tablespace Altered
```

修改控制文件，把数据文件 PRODUCT2 的位置改成 F:\NEWDB\SECOND。

STEP 05 使表空间 product 重新在线。

```
SQL> ALTER TABLESPACE PRODUCT ONLINE;
Tablespace Altered
```

STEP 06 再次查询数据文件的信息。

```
SQL> SELECT file_name,Tablespace_name FROM DBA_DATA_FILES;
```

SQL 语句的执行结果如下所示：

```
FILE_NAME                                          TABLESPACE_NAME
------------------------------------------------------------------------
......
F:\NEWDB\SECOND\PRODUCT2                 PRODUCT
......
```

从输出结果可以看出，数据文件 PRODUCT2 的位置已经从 F:\NEWDB\DATDAT\WM\改成 F:\NEWDB\SECOND。

STEP 07 立即备份数据库。



注：当数据库的物理结构发生变化时，一定要备份数据库。

7.2.3 删除数据文件

要想删除数据文件，数据文件必须为空。

```
SQL> ALTER TABLESPACE product DROP DATAFILE 'F:\NEWDB\DATDAT\WM\PRODUCT4';
Tablespace Altered.
```

其中，DROP DATAFILE 表示删除数据文件；product 是数据文件所属的表空间名；PRODUCT4 是被删除的数据文件。

如果要删除临时文件，可以使用类似于下面的命令：

```
ALTER TABLESPACE lmtemp DROP TEMPFILE '/u02/Oracle/data/lmtemp02.dbf';
```

其中，lmtemp 是临时表空间的名字；lmtemp02.dbf 是被删除的临时文件的名字；TEMPFILE 是关键字，表示删除临时表空间中的临时文件。

也可以使用下面的命令删除临时文件：

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
INCLUDING DATAFILES;
```

7.2.4 列出数据库中的数据文件

数据字典 V$DATAFILE 存放表空间的信息，从该视图中可以知道数据库中有哪些数据文件。

```
SQL> SELECT file#,name FROM v$datafile;
```

SQL 语句的执行结果如下所示：

```
FILE#       NAME
---------------------------------------------------
       1       D:\NEWDB\ZDB\SYSTEM01.DBF
       2       D:\NEWDB\ZDB\SYSAUX01.DBF
…
```

7.3 临时表空间管理

临时表空间，顾名思义，就是临时存放数据的表空间。这些临时数据包括：

- 索引数据
- 临时表数据、临时索引数据
- 临时大对象（LOB）的信息

创建临时表空间的例子如下所示：

```
CREATE TEMPORARY TABLESPACE mytemp TEMPFILE ' F:\NEWDB\ZDB\userdata1'
SIZE 20M
```

本例中创建临时表空间 mytemp，临时数据文件是 userdata1，临时数据文件的大小是 20MB。值得注意的是，创建临时表空间，用到关键字 TEMPORARY 及 TEMPFILE。TEMPFILE 用于指定临时数据文件。有关“临时数据文件”，本章后续小节将有介绍。

知识点索引——排序段（Sort Segment）

临时表空间中含有排序段（Sort Segment），临时数据放在排序段中。临时数据是进行排序时用到的一些数据，这些数据暂时存放在临时段中。一个 SQL 语句可能会使用一个或者多个临时表空间。临时表空间中不能创建永久性的数据库对象（如表、索引）。

一个排序段可以被多个排序操作所共享。

当执行第一次排序时，Oracle 在临时表空间中创建排序段，该排序段不断扩展，直到能够容纳所有的排序数据。但数据库关闭时，临时段被释放。

下面的 SQL 语句将会用到排序段（临时表空间）：

- ✓ CREATE INDEX
- ✓ SELECT ... ORDER BY
- ✓ SELECT DISTINCT ...
- ✓ SELECT ... GROUP BY
- ✓ SELECT ... UNION
- ✓ SELECT ... INTERSECT
- ✓ SELECT ... MINUS



知识点索引——临时数据文件（Temporary Datafiles）

临时数据文件属于临时表空间。临时数据文件和常规的数据文件差不多，只是有下面的一些差异：

- ✓ 临时数据文件总是被设置成 NOLOGGING。
- ✓ 不能使临时数据文件只读。
- ✓ 不能使用 ALTER DATABASE 创建临时数据文件。
- ✓ 介质恢复（Media Recovery）不能恢复临时数据文件。
- ✓ 创建临时数据文件时，Oracle 并不总是给它分配空间，直到需要使用这些空间。
- ✓ 临时数据文件的信息并不是存在于数据字典 DBA_DATA_FILES 或 V$DATAFILE 中。临时数据文件的信息存放在单独的数据字典 DBA_TEMP_FILES 和 V$TEMPFILE 中。



知识点索引——默认临时表空间（Default Temporary Tablespace）

创建用户时，应该给用户指定一个临时表空间，用户的排序段存放在该表空间。如果没有给用户指定临时表空间，Oracle 会自动给这个用户指定一个临时表空间，这个临时表空间叫默认临时表空间（Default Temporary Tablespace）。如果系统表空间（SYSTEM Tablespace）是本地管理的（Locally Managed），则必须定义一个默认临时表空间，SYSTEM 表空间不能作为默认临时表空间；如果系统表空间是字典管理的（Dictionary Managed），则不必定义一个默认临时表空间，SYSTEM 表空间可以作为默认临时表空间，但是，一旦用户使用 SYSTEM 表空间进行排序，Oracle 将会发出警

告。从性能的角度考虑，不应该把 SYSTEM 表空间作为临时表空间。

创建数据库的时候，可以指定默认临时表空间。数据库创建完成以后，我们也可以修改默认的临时表空间。

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tablespace_name;
```

找出默认临时表空间：

```
SQL> SELECT PROPERTY_NAME, PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
  2  PROPERTY_NAME='DEFAULT_TEMP_TABLESPACE';
```

SQL 语句的执行结果如下所示：

```
PROPERTY_NAME                  PROPERTY_VALUE
------------------------------------------------------------------------
DEFAULT_TEMP_TABLESPACE        TEMP
```

由输出不难看出，默认临时表空间是 TEMP。



知识点索引——表空间组（Tablespace Groups）

在 Oracle 11g 以前，一个用户只能使用一个临时表空间，如果临时表空间不足，将导致排序错误。同时，单个表空间的机制影响数据库的性能。Oracle 11g 引入了表空间组，一个用户可以同时使用多个表空间，这样的益处不言而喻，尤其值得注意的是，它使在多个临时表空间上的并行执行成为现实。下面我们将通过一连串的例子讲述表空间组的使用。



创建临时表空间 mytemp1。

```
SQL> CREATE TEMPORARY TABLESPACE mytemp1 TEMPFILE 'C:\ZHAO\VMDATA\
tempfile1' SIZE 10M;
表空间已创建。
```



把临时表空间 mytemp1 加入表空间组 **group3**（如果 group3 不存在，该语句隐含创建表空间组 group3）。

```
SQL> ALTER TABLESPACE mytemp1 TABLESPACE GROUP group3;
表空间已更改。
```



创建表空间 mytemp2，同时把表空间加入表空间组 **group3**。

```
SQL> CREATE TEMPORARY TABLESPACE mytemp2 TEMPFILE 'C:\ZHAO\VMDATA\
tempfile2' SIZE 10M
  2  TABLESPACE GROUP group3;
表空间已创建。
```



查询表空间组的信息。

```
SQL> select * FROM DBA_TABLESPACE_GROUPS;
```

SQL 语句的执行结果如下所示：

```
GROUP_NAME                     TABLESPACE_NAME
------------------------------ ------------------------------
GROUP3                         MYTEMP1
```

```
GROUP3                         MYTEMP2
SQL>
```

从查询结果可以看出，表空间组 **GROUP3** 由两个临时表空间 MYTEMP1、MYTEMP2 组成。

STEP 05 把默认临时表空间改成表空间组 group3。

```
SQL> ALTER DATABASE DEFAULT TEMPORARY TABLESPACE group3;
数据库已更改。
```

STEP 06 查看当前数据库的默认临时表空间。

```
SQL> SELECT PROPERTY_NAME, PROPERTY_VALUE FROM DATABASE_PROPERTIES WHER
  2  PROPERTY_NAME='DEFAULT_TEMP_TABLESPACE';
```

SQL 语句的执行结果如下所示：

```
PROPERTY_NAME                  PROPERTY_VALUE
------------------------------------------------------------------------
DEFAULT_TEMP_TABLESPACE        GROUP3
```

从输出结果可以看出，默认临时表空间已经变成表空间组 **GROUP3**。

STEP 07 把临时表空间 mytemp2 从表空间组 group3 中删除。

```
SQL> ALTER TABLESPACE mytemp2 TABLESPACE GROUP '';
表空间已更改。
```

STEP 08 再次查看表空间的信息。

```
SQL> SELECT * FROM DBA_TABLESPACE_GROUPS;
```

SQL 语句的执行结果如下所示：

```
GROUP_NAME                     TABLESPACE_NAME
------------------------------------------------------------
GROUP3                         MYTEMP1
SQL>
```

查询结果显示，临时表空间 mytemp2 已经从表空间组 group3 中被删除。

视图 DBA_TEMP_FILES 用于描述临时文件的信息。从该视图中，可以知道一个数据库中有哪些临时表空间。

```
SQL> SELECT Tablespace_name,file_name,status
  2  FROM dba_temp_files
  3  ORDER BY Tablespace_name;
```

SQL 语句的执行结果如下所示：

```
TABLESPACE_NAME    FILE_NAME                              STATUS
---------------------------------------------------------------------
MYTEMP1     C:\ZHAO\VMDATA\TEMPFILE1                AVAILABLE
MYTEMP2     C:\ZHAO\VMDATA\TEMPFILE2                AVAILABLE
TEMP            D:\NEWDB\ZDB\TEMP01.DBF             AVAILABLE
```

查询结果显示，数据库中共有三个临时表空间（MYTEMP1、MYTEMP2、TEMP）。

7.4 回滚表空间管理

回滚表空间是表空间的一种类型，用于存放回滚数据。本节的学习重点在于理解回滚数据、回滚段、回滚表空间。

7.4.1 回滚数据（Undo Data）

当用户修改数据的时候，Oracle 会把数据修改以前的值保存起来，这样的数据叫回滚数据（Undo Data）。回滚数据有下面的用途：

- 事务回滚（Transaction Undo），回滚未提交的事务
- 提供读一致性（Read Consistency）
- 数据库恢复（Database Recovery）
- 闪回查询（Flashback Query）

回滚数据一直被保存在回滚段中，直到下面的情况出现：

- 用户输入了 ROLLBACK 命令回滚了这个事务（Transaction）
- 用户输入了 COMMIT 命令提交了这个事务（Transaction）
- 用户（会话）被异常中止（事务被回滚）
- 用户正常退出 Oracle 登录（事务被提交）

7.4.2 读一致性（Read Consistency）

回滚数据为 Oracle 提供读一致性。当一个用户启动一个事务，对数据库进行修改，但数据未提交(Commit)，而正在此时，另外一个用户查询刚才那个用户修改的数据，则 Oracle 会从回滚段中获得查询需要的数据。

例如：有一个学生信息表如图 7-9 所示。

Name	Id	home
李才	111	上海
李丽	112	天津
王军	114	北京
高军	115	上海

图 7-9　学生信息表

Name 表示学生姓名；Id 表示学生编号；home 表示学生的家乡所在地。

如图 7-10 所示，11:01，用户 A 删除家乡在“上海”的学生的信息（李才和高军），但是未提交，这时候，高军和李才的信息已经被复制到回滚段中。11:02，用户 B 要查询这

两个学生的信息，这时候，Oracle 根据回滚段中的回滚数据重构这两个学生的信息，最终，用户 B 查询到这两个学生的信息。11:03，用户 A 提交对两个学生的修改，这两个学生的回滚数据不再保存在回滚段中，这时候，如果用户 B 想再次查询这两个学生的信息，将什么也看不到。

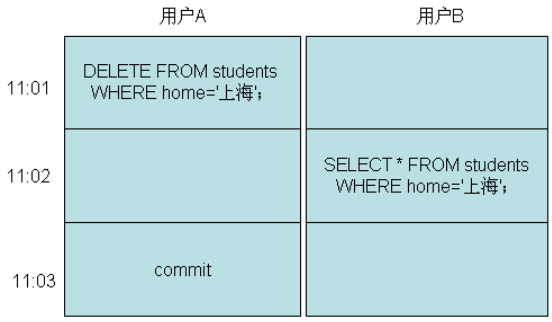


图 7-10　读一致性的例子

7.4.3　数据库恢复（Database Recovery）

回滚数据是实例恢复（Instance Recovery）的重要组成部分。实例恢复是 Oracle 自动执行的。进行实例恢复时，Oracle 应用重做日志文件中提交（Committed）的和未提交（Uncommitted）的事务，把数据库恢复到实例失败时的状态，最后，Oracle 回滚未提交的事务，Oracle 回滚的时候就会用到回滚段中的数据。

7.4.4　闪回查询（Flashback Query）

闪回查询是 Oracle 的新特性，它允许用户查询过去某个点（如时间点）表的状态。这项功能允许 DBA 把表恢复到过去的某个时间点。

7.4.5　事务和回滚数据（Transactions and Undo Data）

启动一个事务的时候，Oracle 会把一个回滚段赋予这个事务。当数据被改变的时候，数据被改变以前的值将被复制到回滚段中，如果想知道哪个回滚段被赋予哪个事务，可以查询数据字典视图 V$TRANSACTION。

回滚段是特殊的段，需要的时候，Oracle 会自动创建回滚段。与其他类型的段一样，回滚段由区组成，区由块组成。回滚段会自动增长（Grow）和收缩（Shrink）。

7.4.6　回滚段（Undo Segment）

回滚段与数据库中的数据段和索引段一样，都用于存放数据，只是回滚段存放的是回

滚数据。回滚段存放在一种特殊的表空间中，这种表空间叫“回滚表空间（Undo Tablespace）”。一个数据库可以有多个回滚表空间，但是只能有一个活动（Active）的回滚表空间。回滚段存放一个或者多个表的回滚数据。

回滚段中的区是循环使用的，当回滚段中的一个区使用完成的时候，Oracle 会使用一个已经存在的区或者分配一个新的区。当所有的区都使用完成的时候，如果第一个区没有被其他活动事务所使用，Oracle 会重新使用第一个区。下面，我们将通过一个例子说明回滚段的使用，如图 7-11 所示。

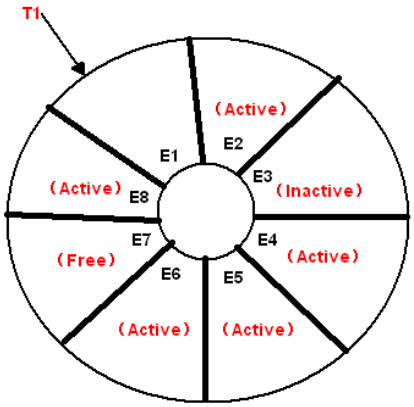


图 7-11　回滚段的原理

事务 T1 开始写区 E1，当 E1 写满的时候，Oracle 检查下一个区，发现区 E2 上有活动的事务（即有未提交的数据）；Oracle 继续检查下一个区，发现区 E3 是不活动的（数据已经提交），Oracle 把回滚数据写到 E3 中，当 E3 写满的时候，Oracle 对后续的区进行检查，发现 E4、E5、E6 中有活动事务，这几个区不能写，Oracle 又继续检查，发现有一个空白的区 E7，Oracle 开始往这个区中写回滚数据，当 E7 写满后，Oracle 继续检查，发现最后一个区 E8 是活动的，这时候，所有的区已经用完，Oracle 循环到区 E1，如果区 E1 可用，Oracle 会重用区 E1；如果区 E1 不可用，Oracle 只能给这个回滚段分配一个新的区。

当用户输入 DML（INSERT、UPDATE、DELETE）后，就开始了一个事务，Oracle 把该事务和一个回滚段进行关联。Oracle 将使用这个回滚段存放这个事务的回滚数据。对表中的任何数据的改变将被记录到回滚段中。当前活动的回滚段的信息，可以通过查询视图 V$ROLLNAME 获得。


```
SQL> SELECT * FROM V$ROLLNAME;
```

SQL 语句的执行结果如下所示：

```
  USN NAME
---------- ------------------------------
         0 SYSTEM
         0 SYSTEM
        11 _SYSSMU11_1249293572$
```

```
        12 _SYSSMU12_1249293572$
        13 _SYSSMU13_1249293572$
        14 _SYSSMU14_1249293572$
        15 _SYSSMU15_1249293572$
        16 _SYSSMU16_1249293572$
        17 _SYSSMU17_1249293572$
        18 _SYSSMU18_1249293572$
        19 _SYSSMU19_1249293572$
        20 _SYSSMU20_1249293572$
11 rows Selected
```

无论哪个用户使用回滚段，所有回滚段的拥有者都是用户 SYS。一个回滚段至少有两个区。因为至少要两个区才能实现回滚段的循环利用。一个回滚段可以有许多区，对于块大小是 8KB 的表空间，一个回滚段默认情况下最多可有 32765 个区。

回滚段有两种管理方式：手动回滚段管理（Manual Undo Management）和自动回滚段管理（Automatic Undo Management）。手动回滚段方式比较烦琐，许多工作都需要 DBA 手动完成，如手动创建回滚段，如图 7-12 所示，手动使回滚段在线（Online）。这些手动的操作增加 DBA 的管理负担，使回滚段的管理变得更加复杂。手动回滚段管理正逐渐被 Oracle 所淘汰。

```
CREATE ROLLBACK SEGMENT rbs01
TABLESPACE users
STORAGE(
INITIAL 100K
NEXT 100K
MINEXTENTS 20
MAXEXTENTS 100);
```

图 7-12　手工创建回滚段

自动回滚段管理（Automatic Undo Management）使 DBA 能够从繁重的工作中解脱出来。自动回滚段管理只需一个回滚表空间。回滚段的创建和释放等工作都由 Oracle 自动完成。DBA 唯一的工作是设置初始化参数 UNDO_MANAGEMENT=AUTO，并且通过初始化参数 UNDO_TABLESPACE 指定数据库使用哪个回滚表空间。目前，自动回滚段管理方式已经得到广泛应用。

7.4.7　回滚表空间（Undo Tablespace）

回滚表空间是一个特殊的表空间，回滚表空间只能用于存放回滚段。不能在回滚表空间创建其他数据库对象（如表、索引）。每个数据库可有零个或者多个回滚表空间。但是，只能有一个活动的回滚表空间。

SYSTEM 回滚段是特殊的回滚段，它存在于 SYSTEM 表空间，SYSTEM 回滚段用于回滚系统事务。非系统事务只能交由非系统回滚段来处理。因此，除了系统回滚表空间外，每个数据库应该另外创建一个回滚表空间。

7.4.8 回滚表空间的操作

前面的几节主要介绍回滚表空间的理论，接下来，我们将介绍回滚表空间的使用。

1．创建回滚表空间

创建回滚表空间的例子如下所示：

```
SQL> CREATE UNDO TABLESPACE myundotbs
  2  DATAFILE 'C:\ZHAO\VMDATA\undo0201.dbf' SIZE 2M REUSE AUTOEXTEND ON;
表空间已创建。
```

其中，CREATE UNDO TABLESPACE 表示创建回滚表空间；DATAFILE 用于指定数据文件；SIZE 用于指定数据文件的大小；REUSE 表示重用已经存在的数据文件；AUTOEXTEND ON 表示使数据文件可以自动扩展。

2．启用回滚表空间自动管理

在 Oracle 9i 及以后版本中，引入了回滚段自动管理的概念。所谓“回滚段自动管理”就是回滚段的创建、删除等工作都由 Oracle 自动完成，不需要 DBA 的干预。回滚段自动管理简化了回滚段的管理，把 DBA 从烦琐的管理工作中解脱出来。

STEP 01 创建回滚表空间（undotbs2）。

```
SQL> CREATE UNDO TABLESPACE undotbs2
  2  DATAFILE 'E:\ORACLE11f\DATDAT\undo2'
  3  SIZE 10M REUSE AUTOEXTEND ON;
Tablespace created
```

其中，undotbs2 是回滚表空间的名字；undo2 是组成回滚表空间的数据文件；SIZE 10M 表明数据文件的大小是 10MB；REUSE 表明如果已经存在名字相同的数据文件，则重用它；AUTOEXTEND ON 把数据文件设置成自动扩展。

STEP 02 关闭数据库。

```
SQL> SHUTDOWN IMMEDIATE;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL>
```

STEP 03 修改初始化参数文件（本例是 INITZDB.ORA），增加两个参数。

```
UNDO_MANAGEMENT= AUTO
UNDO_TABLESPACE = undotbs2
```

其中，UNDO_MANAGEMENT=AUTO 表示启用自动管理的回滚段；UNDO_TABLESPACE = undotbs2 指定回滚表空间，undotbs2 是上面创建的回滚表空间。

STEP 04 创建 SPFILE（可选）。

```
    CREATE    SPFILE=    C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
    FROM PFILE = 'C:\app\Administrator\product\11.1.0\db_3\database\INITZDB.ORA'
```

其中，SPFILEZDB.ORA 是新建的 SPFILE；INITZDB.ORA 是修改过的初始化参数文件（PFILE）。

STEP 05 重新启动数据库。

- 如果执行了 STEP 04，则使用 SPFILE 启动数据库。

```
    SQL>STARTUP;
```



注：这里不需要指定 SPFILE 的名字。

- 如果没有执行 STEP 04，也就是没有创建 SPFILE，则使用初始化参数文件（INITZDB.ORA）启动数据库。

```
    SQL>STARTUP    PFILE='C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA';
```

3．删除回滚表空间

管理员可以删除回滚表空间，但是不能删除数据库当前正在使用的回滚表空间。如果要删除数据库正在使用的回滚段，只能把数据库切换到另外一个回滚表空间。回滚表空间的删除和普通表空间的删除方式相同。

4．往回滚表空间中增加数据文件

往回滚表空间中添加数据文件的例子如下：

```
    SQL> ALTER TABLESPACE myundotbs
      2 ADD DATAFILE 'C:\ZHAO\VMDATA\undo0202.dbf' SIZE 5M AUTOEXTEND ON NEXT 1M
     MAXSIZE UNLIMITED;
    表空间已更改。
```

往表空间 myundotbs 中增加数据文件 undo0202.dbf。

其中，SIZE 表示数据文件的初始大小；AUTOEXTEND ON 表示数据文件被设置成自动扩展；NEXT 1M 表示每次以 1MB 为单位进行扩展；MAXSIZE UNLIMITED 表示该文件可以无限制扩展。

5．切换数据库当前的回滚表空间

我们可以创建多个回滚表空间，但是，同时只能有一个表空间被使用，可以很方便地从一个表空间切换到另外一个表空间。

STEP 01 显示数据库当前正在使用的回滚表空间。

```
    SQL> SHOW PARAMETER  UNDO_TABLESPACE
```

SQL 语句的执行结果如下所示：

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------
undo_tablespace                       string      UNDOTBS1
SQL>
```

数据库当前正在使用的回滚表空间是 UNDOTBS1。



把数据库正在使用的回滚表空间切换成 myundotbs。

```
SQL> ALTER SYSTEM SET UNDO_TABLESPACE = myundotbs scope=both;
系统已更改。
```

scope=both 表示这种修改是永久的，并且立即生效！

初始化参数 UNDO_TABLESPACE 控制数据库正在使用的回滚表空间。



查看系统正在使用的回滚表空间。

```
SQL> SHOW PARAMETER  UNDO_TABLESPACE
```

SQL 语句的执行结果如下所示：

```
NAME                                 TYPE        VALUE
-------------------------------- ----------- -------------------------
undo_tablespace                       string      MYUNDOTBS
```

从输出结果可以看出，我们已经成功地把数据库当前的回滚表空间从 UNDOTBS1 切换到 MYUNDOTBS。当然，我们也可以切换回来。

6．查询数据库中的回滚表空间

```
SQL> SELECT TABLESPACE_NAME
  2  FROM DBA_TABLESPACES
  3  WHERE contents='UNDO';
```

SQL 语句的执行结果如下所示：

```
TABLESPACE_NAME
------------------------------
UNDOTBS1
MYUNDOTBS
```


本例中，数据库共有两个回滚表空间：UNDOTBS1、MYUNDOTBS。要想知道哪个表空间正在被数据库使用，请执行 SHOW PARAMETER UNDO_TABLESPACE 命令。



本章总结

本章详细介绍了数据库的物理结构和逻辑结构，并重点介绍了各种类型的表空间。表空间的管理其实很简单，但是，如果想站在性能的角度去看待表空间，需要对表空间的原理进行深入的理解。对做开发的朋友来说，也许了解表空间的使用已经足矣！

第 8 章　简化存储管理之——自动存储管理（ASM）

自动存储管理（Automated Storage Management，ASM）是 Oracle 10g 推出的新功能。它实质上是一个卷管理器，用于替代操作系统所提供的逻辑卷管理器（LVM），对磁盘进行管理。ASM 装载磁盘组，并使 ASM 文件（ASM files）对数据库实例有效，即使数据库实例能够访问 ASM 文件。它同时提供冗余对数据进行保护。ASM 使管理员从管理成百上千数据文件的琐碎事务中解脱出来。

8.1　ASM 实例（ASM Instances）

ASM 实例，其实就是一个实例（和 Oracle 数据库实例类似），它有自己的内存（System Global Area，SGA）和后台进程（Background Processes）。它的任务是加载磁盘组（Disk Groups），使 ASM 文件（ASM files）有效，以便数据库实例（Oracle Database Instance）能够访问磁盘组中的数据。数据库实例（Oracle Database Instance）和 ASM 实例是两种不同的实例，这里要学会区别开。

8.2　ASM 磁盘组（ASM Disk Groups）

ASM 磁盘组其实是一个虚拟的概念，一个 ASM 磁盘组由多个磁盘组成（如图 8-1 所示）。一个 **ASM 文件**只能存放在一个磁盘组中，一个文件所获得的空间要从磁盘组中进行划分。每个磁盘组都包含 **ASM 元数据（ASM Metadata）**。

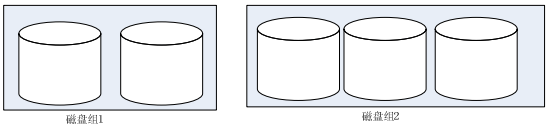


图 8-1　磁盘和磁盘组的关系

ASM 元数据（ASM Metadata）

ASM 元数据驻留在磁盘组（Disk Groups）中，Oracle 使用这些信息管理磁盘组。ASM 元数据包括：

- 一个磁盘属于哪个磁盘组
- 磁盘组上的有效空间
- 一个磁盘组中有哪些文件
- 区（Extent）的位置

ASM 磁盘（ASM Disk）

ASM 磁盘是真正存放数据的地方，一个 ASM 磁盘可以是：

- 磁盘阵列上的一个磁盘或者分区
- 一整块磁盘或者来自一个磁盘上的分区
- 逻辑卷（Logical Volumes）
- 网络文件系统（Network-attached Files，NFS）

分配单元（Allocation Unit，AU）

磁盘组中的每个磁盘会被分成多个单元（Allocation Units，AU），AU 是磁盘组中空间分配的单位。在创建磁盘组时，可以指定 AU 的大小。AU 的大小可以是 1MB、2MB、4MB、8MB、16MB、32MB 或者 64MB。AU 是 ASM 存储中结构最小的元素。

8.3 ASM 文件（ASM Files）

存放在 ASM 磁盘组中的文件（ASM Disk Group）叫 ASM 磁盘文件，一个 ASM 文件由多个区（Extent）组成（如图 8-2 所示）。一个 ASM 磁盘文件只能存放在一个磁盘组中。可以存放在磁盘组中的文件有：

- 控制文件（Control files）
- 数据文件（Datafiles）、临时文件（Temporary Datafiles）和数据文件副本（Datafile Copies）
- 服务器参数文件（SPFILE）

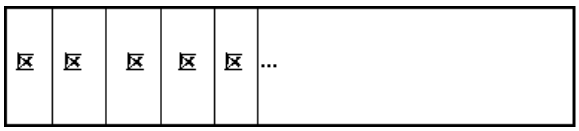


图 8-2　区和 ASM 文件的关系

- 重做日志文件（Online Redo Logs）、归档日志文件（Archive Logs）和闪回日志文件（Flashback Logs）
- RMAN 生成的备份（RMAN Backups）

- 灾难恢复配置（Disaster Recovery Configurations）
- Change Tracking Bitmaps（改变跟踪位图）

ASM 自动产生 ASM 文件的文件名，我们也可以给 ASM 文件指定一个容易记忆的别名。

8.4 区（Extents）

ASM 磁盘组上创建文件时，最小的可寻址单元是区（ASM 区和数据库的区概念不相同）。在 Oracle 数据库 11g 中，区大小不再等于 AU 大小。每个区驻留在磁盘组中一个单独的磁盘，每个区由多个 AU 组成（如图 8-3 所示）。区的大小可变，区大小从 1MB 开始，文件达到某个阈值时，区大小增加至 4MB，然后是 16MB，最后当达到某个阈值后，区大小为 64MB。你不必担心大小；ASM 实例会自动分配合适的区大小。由于较少数量的区即可容纳大量数据，因此可以大大减少共享池中的区总数，从而将性能提高数倍。

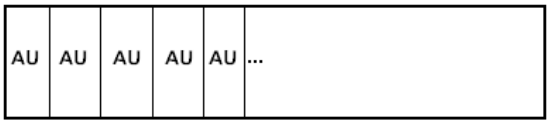


图 8-3　区（Extent）和 AU 的关系

8.5 镜像和失败组（Mirroring and Failure Groups）

ASM 通过镜像来实现对数据的保护，Oracle 提供三种冗余机制（Redundancy Level Controls）来保护数据：

- 常规冗余（Normal Redundancy），只有两份相同的区（Extent），一份存放在一个磁盘组中，另外一份存放在另一个磁盘组中（如图 8-4 所示）。常规冗余只能容忍一个磁盘组的损坏。

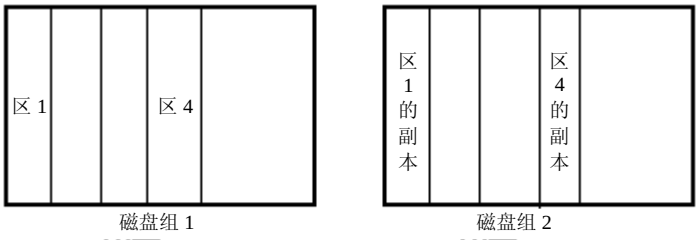


图 8-4　常规冗余（Normal Redundancy）

- 高度冗余（High Redundancy），只有 3 份相同的区（Extent），分别放在 3 个磁盘组中。其中两个磁盘组损坏，也不会导致数据的丢失（如图 8-5 所示）。
- 外部冗余（External Redundancy），指 ASM 不提供冗余保护，冗余保护由第三方的硬件或者软件完成。



见图 8-5　高度冗余（High Redundancy）

在一组磁盘内，ASM 会把每个区（Extent）尽量分布在不同的 ASM 磁盘（如图 8-6 所示）。

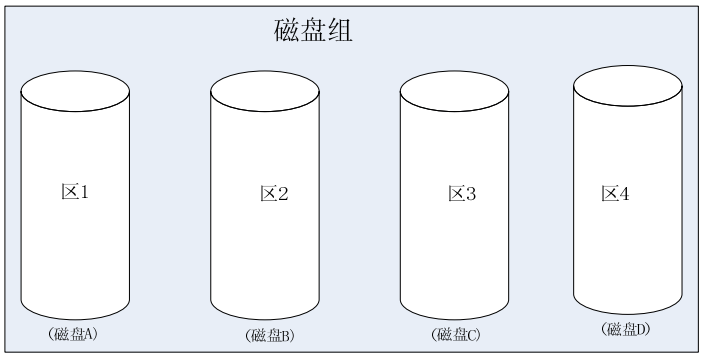


图 8-6　条块化（Striping）

本例中，数据被分配到 4 块磁盘（磁盘 A、磁盘 B、磁盘 C、磁盘 D），当要读取数据时，Oracle 会并行读取 4 块磁盘中的数据。这样的机制加快了数据的读取速度。

8.6　ASM 使用全程

若要使用 ASM，必须提前配置好 ASM 实例！

8.6.1　配置 ASM 实例

配置 ASM 实例需要经历两个阶段：准备磁盘和创建 ASM 实例（本文以 Windows XP 为例）。

1．准备磁盘

若想使用磁盘，首先必须划分磁盘（分区），然后使用 asmtoolg 来标记磁盘。标记磁盘的目的是使 ASM 可以识别这些磁盘（分区）。

📖 **分区**

STEP 01 单击“开始”→“控制面板”→“性能和维护”→“管理工具”→“计算机管理”→“磁盘管理”，出现如图 8-7 所示的未分配空间。

单击鼠标右键，选择“新建磁盘分区”，出现如图 8-8 所示的界面。



图 8-7　未分配空间

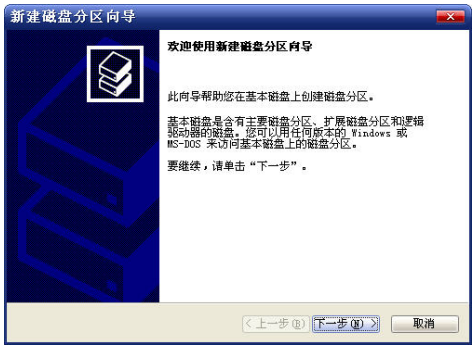


图 8-8　磁盘分区首页

单击 下一步(N) > 按钮，出现如图 8-9 所示的界面。

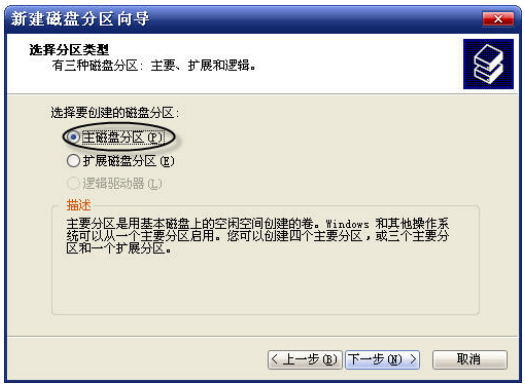


图 8-9　选择分区类型

STEP 02 选择分区类型。

这里要选择“主磁盘分区”，否则 asmtoolg 工具不能识别创建的分区。选中“主磁盘分区”，单击 下一步(N) > 按钮，出现如图 8-10 所示的界面。

STEP 03 指定分区大小。

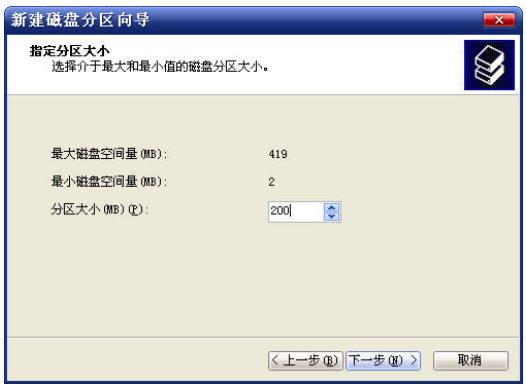


图 8-10　指定分区大小

输入分区大小，单击 下一步(N) > 按钮，出现如图 8-11 所示的界面。

STEP 04　指派驱动器号和路径。

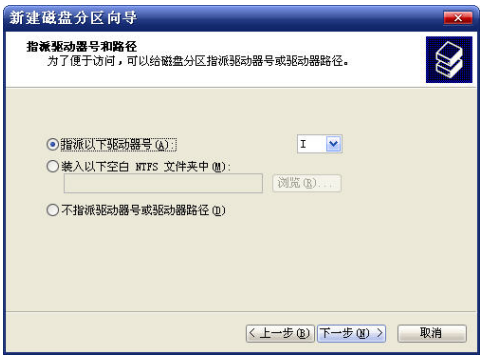


图 8-11　指派驱动器号和路径

接受默认值，单击 下一步(N) > 按钮，出现如图 8-12 所示的界面。

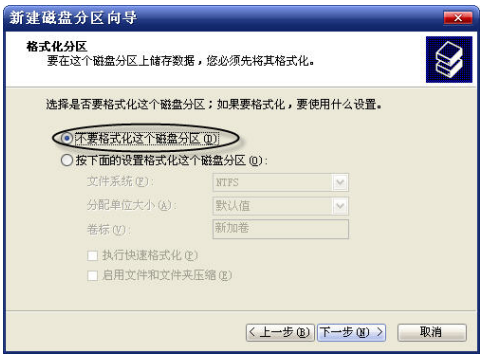


图 8-12　格式化分区

STEP 05　格式化分区。

如果对分区进行格式化，asmtoolg 可以看到分区，但是不能对分区进行操作。选择“不要格式化这个磁盘分区”，然后单击 下一步(N) > 按钮，出现如图 8-13 所示的界面。

STEP 06 完成分区创建。

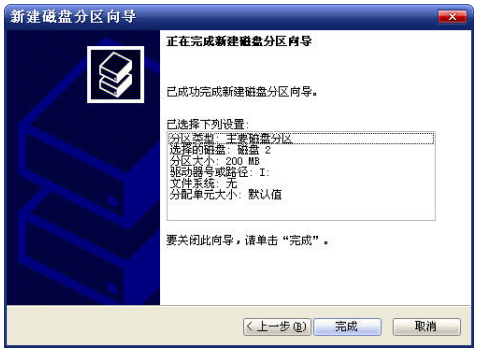


图 8-13　完成分区创建

单击 完成 按钮，完成分区创建。

使用 asmtoolg 标记磁盘

STEP 01 启动 asmtool。

在操作系统命令行中输入命令 asmtoolg，启动 asmtool，启动后的界面如图 8-14 所示。

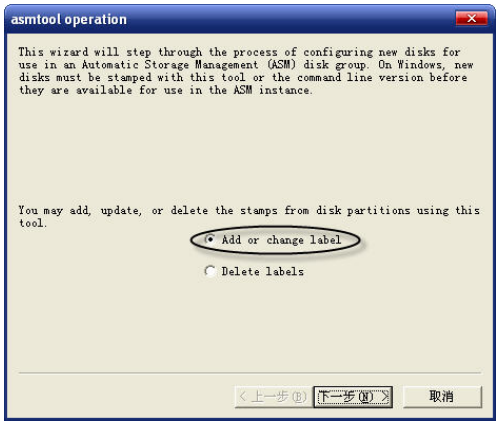


图 8-14　asmtool 的主界面

知识点索引——磁盘标记工具 asmtool

要让 ASM 能够识别磁盘，需要标记磁盘，可以使用两个小工具（实质上是一个）来标记磁盘。一个工具是 asmtool，基于命令行的；另一个工具是 asmtoolg，基于图像界面的。标记后的磁盘方可被 ASM 识别。

STEP 02 标记磁盘。

选项“Delete labels”表示删除标记，“Add or change label”表示标记磁盘。选择“Add or change label”，单击 下一步(N) > 按钮，出现如图 8-15 所示的界面。

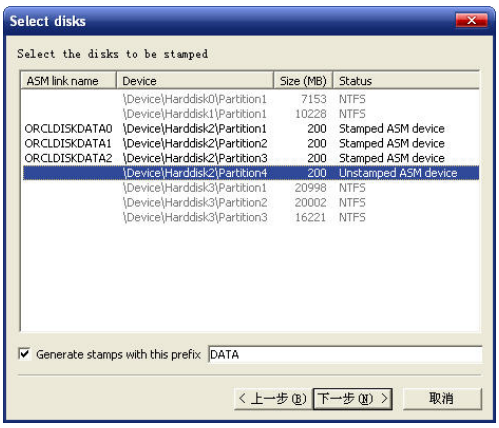


图 8-15　标记磁盘

图 8-15 中列出了已经被标记的磁盘（Stamped ASM device）和未被标记的（Unstamped ASM device）磁盘 Partition4。

“Generate stamps with this prefix”指定链接名（Link Name）的前缀。通常接受默认值 DATA。这样生成的链接名的格式是 ORCLDISKDATAn。

选中分区 Partition4，单击 下一步(N) > 按钮，出现如图 8-16 所示的界面。

单击 下一步(N) > 按钮，出现如图 8-17 所示的界面。



图 8-16　确认选中的分区

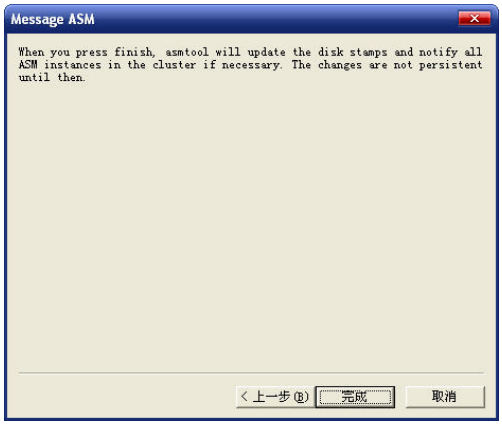


图 8-17　分区标记完成

单击 完成 按钮，分区标记完成。已经被标记的分区可被 ASM 识别。

2．创建 ASM 实例

磁盘准备就绪，接下来，我们将创建 ASM 实例。本文将使用 DBCA 创建 ASM 实例。用 DBCA 创建 ASM 实例简单非常快捷！创建 ASM 实例的过程说明如下。

STEP 01　启动群集同步服务。

群集同步服务（Cluster Synchronization Services，CSS）负责 ASM 和数据库实例（Database Instances）之间的通信，因此，在启动 DBCA 之前，必须启动 CSS，否则就会

提示如图 8-18 所示的信息。

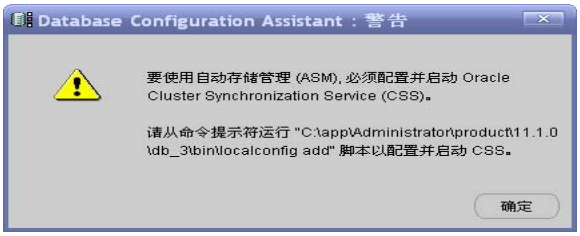


图 8-18　警告

ⓐ 以系统管理员的身份登录操作系统。
ⓑ 进入路径 ORACLE_HOME\bin。
ⓒ 用命令 localconfig add 创建并启动 CSS。

```
C:\Documents and Settings\Administrator>localconfig add
Step 1:  creating new OCR repository
Successfully accumulated necessary OCR keys.
Creating OCR keys for user 'administrator', privgrp ''..
Operation successful.
Step 2:  creating new CSS service
successfully created local CSS service
successfully added CSS to home
```

这个命令创建并启动 CSS。

STEP 02 启动 DBCA，如图 8-19 所示。



图 8-19　DBCA 主界面

单击 下一步(N) > 按钮，出现如图 8-20 所示的界面。

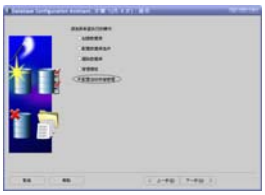


图 8-20　数据库操作选择

选择“配置自动存储管理”，单击 下一步(N) > 按钮，出现如图 8-21 所示的界面。



图 8-21　密码设置

STEP 03 sys 的密码设置。

在该界面中设置 ASM 管理员 sys 的密码。特别值得注意的是，这里的 sys 用户和数据库实例的 sys 用户不是同一个用户，这里的 sys 用户是 ASM 实例的用户，我们以前常说的是数据库实例的用户，它们是两个不同的用户。它们相同的地方是都用来管理实例，只是其中一个用户用来管理数据库实例，另外一个用户用来管理 ASM 实例。

输入 sys 的密码，本例中是 zero。单击 ASM 参数... 按钮，出现如图 8-22 所示的界面。

STEP 04 设置 ASM 参数。



图 8-22　设置 ASM 参数

这里可以设置 ASM 的诸多参数。ASM_DISKSTRING 参数的设置尤为重要，因为该参数设置了磁盘的搜索路径。它可以被设置成类似于/dev/rdsk/*或者/dev/rdsk/*disk3、/dev/rdsk/*disk4 的形式。其中星号（*）标识通配符。对于参数 ASM_DISKSTRING，不同的操作系统有不同的格式。只有满足参数 ASM_DISKSTRING 指定格式的磁盘才能被搜索到。如果 ASM 实例已经创建，我们可以使用类似于这样的命令“ALTER SYSTEM SET ASM_DISKSTRING='\\.\ORCLDISK*' scope=both”修改 ASM_DISKSTRING 的值。

本例中，输入“\\.\ORCLDISK*”（如图 8-22 所示），单击“关闭”按钮，回到如图 8-21 所示的界面。单击 下一步(N) 按钮，出现如图 8-23 所示的界面。

STEP 05 ASM 实例创建进行中。



图 8-23　ASM 实例创建

STEP 06 ASM 实例创建完成。

ASM 实例创建结束后，出现如图 8-24 所示的界面。



图 8-24　创建完成

单击 完成(F) 按钮，出现如图 8-25 所示的界面，询问是否需要执行其他操作，单击 否 按钮，ASM 实例创建彻底完成。



图 8-25　是否执行其他操作

实例创建完成后，我们可以到 Windows 的“服务”中查看 ASM 实例创建是否真的成功了。如果成功，将会看到如图 8-26 所示的条目。



图 8-26　ASM 服务

8.6.2　ASM 实例的连接

ASM 实例的连接和数据库实例的连接很相似，详细过程说明如下。

STEP 01　设置环境变量 ORACLE_SID。

无论在 Windows 还是 UNIX 中，都需要设置环境变量 ORACLE_SID。ORACLE_SID 的值通常是+ASM。在 Windows 平台，只需执行命令 set ORACLE_SID=+ASM 即可。

STEP 02　启动 SQL*Plus，登录 ASM 实例。

```
SQLPLUS /NOLOG
SQL> CONNECT sys AS SYSASM
Enter password:
Connected to an idle instance.
```

这里要以 SYSASM 的身份登录 ASM。SYSASM 是一个系统权限，拥有此权限的用户才有资格管理 ASM 实例。输入密码 zero（该密码是在本书 8.1.1 节中进行设置的）。

8.6.3　ASM 的启动

ASM 的启动过程很简单，具体说明如下。

STEP 01　以 SYSASM 的身份登录 ASM。

```
C:\Documents and Settings\Administrator>sqlplus
SQL*Plus: Release 11.1.0.6.0 - Production on 星期五 7 月 31 23:19:35 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
请输入用户名:  sys as sysasm
输入口令:
已连接到空闲例程。
```

STEP 02　启动 ASM 实例。

```
SQL> startup
ASM 实例已启动
Total System Global Area  535662592 bytes
Fixed Size                 1334380 bytes
Variable Size            509162388 bytes
ASM Cache                 25165824 bytes
```

8.6.4 ASM 的关闭

ASM 的关闭过程说明如下。

STEP 01 以 SYSASM 的身份登录 ASM 实例。

```
C:\Documents and Settings\Administrator>sqlplus
SQL*Plus: Release 11.1.0.6.0 - Production on 星期五 7月 31 23:05:19 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
请输入用户名:  sys as sysasm
输入口令:
连接到:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
```

STEP 02 关闭 ASM 实例。

```
SQL> SHUTDOWN IMMEDIATE;
ASM diskgroups dismounted
ASM instance shutdown
```

知识点索引——ASM 的关闭方式

和关闭数据库实例相似，ASM 也有三种关闭方式：NORMAL、IMMEDIATE（TRANSACTIONAL）、ABORT。

- ✓ NORMAL：该方式会等待所有的 SQL 语句执行完成以及所有的用户退出。
- ✓ IMMEDIATE（TRANSACTIONAL）：等待所有的 SQL 语句执行完成，但是不等待用户退出。
- ✓ ABORT：不做任何等待，立即关闭 ASM 实例，下次 ASM 实例启动时，需要执行实例恢复。如果在 ASM 实例上使用 SHUTDOWN ABORT 命令，则将最终迫使所有相关的数据库执行 SHUTDOWN ABORT。

8.6.5 创建磁盘组

创建磁盘组的步骤说明如下。

STEP 01 确认已经发现的磁盘。

如果 ASM 发现了磁盘，这些磁盘的信息将被列在视图 V$ASM_DISK 中。因此，查询 V$ASM_DISK 可以知道已经发现的磁盘。

```
SELECT name, header_status, path FROM V$ASM_DISK;
```

SQL 语句的执行结果如下所示：

```
NAME        HEADER_STATUS       PATH
--------- ------------- --------------------
            CANDIDATE           \\.\ORCLDISKDATA0
```

```
                CANDIDATE          \\.\ORCLDISKDATA1
                CANDIDATE          \\.\ORCLDISKDATA2
                CANDIDATE          \\.\ORCLDISKDATA3
DISK04          MEMBER             \\.\ORCLDISKDATA4
```

如果磁盘的状态（HEADER_STATUS）是 MEMBER，表示该磁盘已经赋予一个磁盘组；如果磁盘的状态是 CANDIDATE 或者 PROVISIONED，表示该磁盘未赋予任何磁盘组。只有未赋予任何磁盘组的磁盘才能被用于创建磁盘组。

上面的输出显示，ORCLDISKDATA0、ORCLDISKDATA1、ORCLDISKDATA2、ORCLDISKDATA3 都没有赋予任何磁盘，即它们不属于任何磁盘组。



创建磁盘组 DG1。

```
SQL> CREATE DISKGROUP dg1 NORMAL REDUNDANCY
  2  FAILGROUP controller1 DISK
  3  '\\.\ORCLDISKDATA0' NAME DISK0,
  4  '\\.\ORCLDISKDATA1' NAME DISK1
  5  FAILGROUP controller2 DISK
  6  '\\.\ORCLDISKDATA2' NAME DISK2,
  7  '\\.\ORCLDISKDATA3' NAME DISK3
  8  ATTRIBUTE 'AU_SIZE'='4M'
  9  /
Diskgroup created.
SQL>
```

上面的 SQL 语句创建磁盘组 DG1，冗余级别是 NORMAL REDUNDANCY（有关冗余级别的介绍，**请参考本书 8.5 节的内容**），磁盘组由两个失败组（controller1 及 controller2）组成。失败组 controller1 有两块磁盘（ORCLDISKDATA0、ORCLDISKDATA1）；失败组 controller2 有两块磁盘（ORCLDISKDATA2、ORCLDISKDATA3）。DISK0、DISK1、DISK2、DISK3 是由用户任意指定的名字，每个名字对应一块磁盘（分区）。AU 的大小是 4MB。



确认创建的磁盘组。

```
SQL> SHOW PARAMETER asm_diskgroups
```

SQL 语句的执行结果如下所示：

```
NAME          TYPE        VALUE
------------------ ------------------- --------------------
asm_diskgroups    string        NULL, DG1
SQL>
```

从输出可以看出，磁盘组 DG1 已经存在。



8.6.6 在磁盘组上创建表空间

在磁盘组上创建表空间和创建普通表空间没有太大差别，只是在指定数据文件的时候

稍微有些不同。

STEP 01 创建表空间 myspace。

```
SQL> CREATE TABLESPACE myspace DATAFILE '+DG1' SIZE 20M;
表空间已创建。
```

本例在磁盘组 DG1 上创建表空间 myspace，组成 myspace 的数据文件的大小是 20MB。值得注意的是，在指定磁盘组时，前面需要加号（+）。

STEP 02 在表空间上创建一个表。

```
SQL> CREATE TABLE kk
  2  (a INT)
  3  TABLESPACE myspace;
表已创建。
```

从这里可以看出，在磁盘组上创建的表空间可以正常使用。



8.7 ASM 磁盘组的维护

■ 往磁盘组中添加磁盘。

```
ALTER DISKGROUP dg1 ADD DISK
'/devices/diska5' NAME diska5,
'/devices/diska6' NAME diska6
```

往磁盘组 dg1 中添加磁盘/devices/diska5 与/devices/diska6。

■ 删除磁盘。

```
ALTER DISKGROUP dg1 DROP DISK diska5;
```

从磁盘组 dg1 中删除磁盘 diska5。

■ 删除磁盘组。

```
DROP DISKGROUP dg1;
```

删除磁盘组 dg1。

■ 查看 ASM 中有哪些磁盘。

执行下面的 SQL 语句：

```
SQL>  SELECT  D.name,G.name  DISK_GROUP,D.FAILGROUP  FROM  V$ASM_DISK
D,V$ASM_DISKGROUP G WHERE D.GROUP_NUMBER=G.GROUP_NUMBER
         2  ORDER BY DISK_GROUP,D.FAILGROUP;
```

SQL 语句的执行结果如下所示：

```
NAME            DISK_GROUP  FAILGROUP
------------------------------------------------------------
ORCLDISKDATA0   DG1         CONTROLLER1
```

```
ORCLDISKDATA1   DG1         CONTROLLER1
ORCLDISKDATA2   DG1         CONTROLLER2
```

输出显示，ASM 实例中共有 3 个磁盘（ORCLDISKDATA0、ORCLDISKDATA1、ORCLDISKDATA2）。

■ 查看 ASM 中有哪些磁盘组。

```
SQL> SELECT NAME FROM V$ASM_DISKGROUP;
```

SQL 语句的执行结果如下所示：

```
NAME
------------------------------------
DG1
```

从查询结果可以看出，ASM 中共有一个磁盘组。



本章总结

ASM 是 Oracle 公司引入的新技术，用于简化磁盘的管理。如果在生产系统中部署 ASM，最好对其进行性能测试。其实，第三方的硬件或者软件也提供类似于 ASM 的功能。在创建分区时，要按照本章的步骤进行，否则工具 asmtoolg 很可能找不到分区。

第 9 章　为数据库优化打基础之——Oracle 内存管理

前文中我们介绍了实例。实例由内存和后台进程组成。本章将重点介绍 Oracle 的内存结构。



9.1　Oracle 内存介绍

Oracle 的内存存放下面的内容：

- 执行的程序代码。
- 连接的会话信息。
- 程序执行期间所需的信息。
- 缓存的数据。
- Oracle 进程间通信需要的共享信息（如锁的信息）。

Oracle 的基本内存结构分成 3 种：

- 系统全局区（System Global Area，SGA），SGA 被所有的服务器进程（Server Processes）和后台进程（Background Processes）所**共享。**
- 程序全局区（Program Global Area，PGA），对服务器进程和后台进程来说，这部分内存是私有（Private）的、不被共享的。每个进程都有一个 PGA。
- 软件代码区（Software Code Area），软件代码区用于存放正在运行的代码或者能被运行的代码。Oracle 数据库代码（Oracle Database Code）就存储在该区域。

如图 9-1 所示，Oracle 的内存结构由 SGA 和 PGA 组成，SGA 又由数据库高速缓冲区（Database Buffer Cache）、日志缓冲区（Redo Log Buffer）、共享池（Shared Pool）、Java 池（Java Pool）、大池（Large Pool）、流池（Streams Pool）等组成。



9.1.1　系统全局区（SGA）

系统全局区是共享的内存结构，这个内存结构用于存放数据和数据库实例（Oracle

Database Instance）的控制信息。多个用户可以同时连接到同一个实例（Instance），这些用户可以共享 SGA 中的数据，因此，系统全局区又叫“共享全局区（Shared Global Area）”。

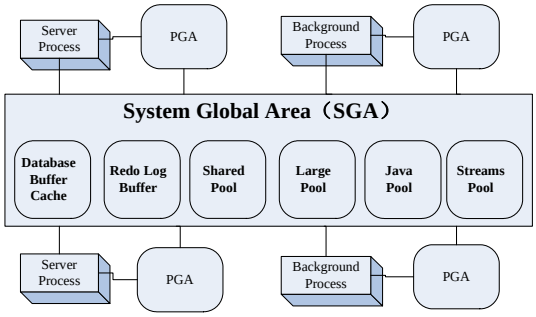


图 9-1　Oracle 内存结构

SGA 和 Oracle 进程都是实例的组成部分，当启动一个实例时，Oracle 会分配 SGA；当关闭一个实例时，Oracle 会把分配给这个实例的 SGA 释放，返回给操作系统。每个实例有一个 SGA。

SGA 是可读写的，所有连接到相同实例的用户都可以读取 SGA 中的信息，但是，只有几个进程可以写 SGA。SGA 主要由下面的结构组成：

- 数据库高速缓冲区（Database Buffer Cache）。
- 重做日志缓冲区（Redo Log Buffer）。
- 共享池（Shared Pool）。
- 大池（Large Pool）。
- Java 池（Java Pool）。
- 流池（Streams Pool）。

SGA 的每部分结构都是为了满足不同的需求。

在 SGA 中，有一部分内存区域用于存放数据库和实例的状态信息（这部分信息不包括用户数据），我们把这部分内存叫**“固定 SGA（Fixed SGA）”**。

1．数据库高速缓冲区（Database Buffer Cache）

数据库高速缓冲区是 SGA 的组成部分，它用于存放从数据文件中读取的数据块。所有连接到相同实例的用户都可以共享这些数据。

2．重做日志缓冲区（Redo Log Buffer）

重做日志缓冲区是 SGA 的组成部分。它是一个循环使用的缓冲区。日志缓冲区用于存放日志条目（Redo Entries），日志条目是有关对数据库的改变（INSERT、UPDATE、DELETE、CREATE、ALTER、DROP）。日志条目可以用于数据库恢复。

当用户对数据库进行改变时，就会产生日志条目，Oracle 把日志条目放到日志缓冲区中。Oracle 的后台进程 LGWR 负责把日志条目写到磁盘上的重做日志文件中（Online Redo Log Files）。

3．共享池（Shared Pool）

共享池也是 SGA 的组成部分，用于存放 SQL 语句、PL/SQL 代码、数据字典、资源锁及其他控制信息。共享池包含库缓冲区（Library Cache）、字典缓冲区（Dictionary Cache）、结果缓冲区（Result Cache）、并行执行消息（Parallel Execution Messages）用到的缓冲区、控制结构占用的缓冲区。共享池的构成如图 9-2 所示。

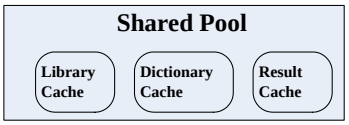


图 9-2　共享池的结构

📖 库缓冲区（Library Cache）

库缓冲区用于存放解析过并执行过的 SQL 和 PL/SQL 代码。当用户发送一条 SQL 或 PL/SQL 代码到数据库时，Oracle 首先对这条语句进行语法解析（Parse），然后将解析后的结果（解析树、执行计划）保存在库缓冲区。如果不同的用户执行相同的 SQL 语句，这些用户可共享相同的解析树（Parse Tree）和执行计划（Execution Plan），这将加快 SQL 语句的执行速度。

库缓冲区包含共享 SQL 区（Shared SQL Areas）、私有 SQL 区（Private SQL Areas）、PL/SQL 过程和包（PL/SQL Procedures And Packages）、控制结构（如锁）。共享 SQL 区可被所有的用户所共享。

- 共享 SQL 区（Shared SQL Area）

共享 SQL 区用于存放语句的解析树和执行计划。当用户发送的一条指令（SQL、PL/SQL）且被解析以后，Oracle 从共享池中分配一定的内存用于存放这条指令的解析树和执行计划。如果另外一个用户执行相同的 SQL 语句，则这个用户可以利用共享 SQL 区中的解析树和执行计划，而不必对这条 SQL 语句进行重新解析，这可以提高 SQL 语句的执行效率。Oracle 用于存放 SQL 语句的内存大小由 SQL 语句的复杂程度决定。

- 私有 SQL 区（Private SQL Area）

SQL 语句的一部分信息存放在共享 SQL 区，另外一部分信息存放在私有 SQL 区中。存放在私有 SQL 区中的这部分信息只被执行这条 SQL 语句的会话（Session）所独有，其他会话不能共享这部分信息。这部分信息包括 SQL 语句的绑定变量、环境和会话参数，还包括运行 PL/SQL 程序单元（PL/SQL Program Unit）所需的局部变量（Local Variable）、全局变量（Global Variable）和包变量（Package Variable）。私有 SQL 区只在共享服务器模式（Shared Server Mode）存在。（有关共享服务器模式的信息，**请参考本书 2.2.3 节的内容**）。

📖 字典缓冲区（Dictionary Cache）

当我们访问数据库的时候，Oracle 会频繁地访问数据字典（Dictionary）。数据字典存放在磁盘上的 SYSTEM 表空间中。为了提高性能，Oracle 把访问频繁的数据字典信息（表、

视图）缓存到数据字典缓冲区中。Oracle 利用数据字典缓冲区中的信息对新输入的 SQL 语句进行解析。一句话，字典缓冲区是存放数据字典的内存区域。

数据字典信息存放在内存中的两个地方：一是数据字典缓冲区，二是库缓冲区。数据字典缓冲区以行（Row）的形式存放数据，而其他缓冲区以块（Block）的形式存放数据，因此，数据字典缓冲区又叫行缓冲区（Row Cache）。所有的用户进程共享数据字典缓冲区和库缓冲区中的数据字典信息。

由于数据字典被频繁访问，字典缓冲区应该足够大以能够容纳我们需要的数据字典信息。字典缓冲区的大小将影响数据库的整体性能！

有关数据字典的信息，**请参考本书 1.6 节的内容**。

📖 **结果缓冲区（Result Cache）**

结果缓冲区存放两种类型的数据：SQL 语句的查询结果（SQL Query Result）、PL/SQL 函数的执行结果（PL/SQL Function Result）。

4．大池（Large Pool）

大池是 Oracle 可选的内存结构，它主要用于大范围的内存需求和分配。下面的几种情况会用到大池：

- 数据库的备份和恢复（Backup、Restore）。
- 并行查询（Parallel Query）。
- 共享服务器模式下的会话内存（Session Memory）。

5．Java 池（Java Pool）

Java 池主要用于支持在数据库中运行 Java 代码。

6．流池（Streams Pool）

流池的主要用途是加强对流（Oracle Streams）的支持。流池存放队列信息（Queue Messages）。如果没有特别指定，流池的尺寸是 0，流池的大小伴随流（Oracle Streams）的需求动态改变。

9.1.2 程序全局区（PGA）

每个服务器进程（Server Process）和后台进程（Background Process）都有一个程序全局区（Program Global Area，PGA），程序全局区用于处理 SQL 语句和容纳会话（Session）信息。通常 PGA 由会话内存区（Session Memory）和私有 SQL 区（Private SQL Area）两部分组成（如图 9-3 所示）。

1．会话内存区（Session Memory）

会话内存区（Session Memory）存放会话变量及其他和会话相关的信息。在共享服务器模式（Shared Server Mode）下，会话内存区（Session Memory）是共享的，而不是私有的。

2．私有 SQL 区（Private SQL Area）

私有 SQL 区（Private SQL Area）存放绑定变量（Bind Variable）的值、查询执行的状态信息、查询执行工作区（Query Execution Work Areas）及游标（Cursor）的信息。每个正在执行 SQL 语句的会话都有一个私有 SQL 区（Private SQL Area）。多个私有 SQL 区（Private SQL Area）对应一个共享 SQL 区（Shared SQL Area），私有 SQL 区和共享 SQL 区的关系如图 9-4 所示。

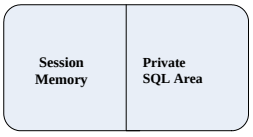


图 9-3　PGA 的组成

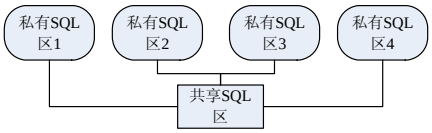


图 9-4　共享 SQL 区和私有 SQL 区的关系

🕮 查询执行工作区（Query Execution Work Areas）

下面的操作会用到“查询执行工作区（Query Execution Work Areas）”：

- 涉及排序的操作（如 Order By、Group By、Rollup、Window Function）
- 哈希连接（Hash-Join）
- 位图合并（Bitmap Merge）
- 位图创建（Bitmap Create）

🕮 私有 SQL 区（Private SQL Area）的位置

私有 SQL 区（Private SQL Area）的位置根据服务器模式的不同而不同。如果请求的是专有服务器模式（Dedicated Server Mode）的连接，则私有 SQL 区（Private SQL Area）位于 PGA 中（如图 9-5 所示）。

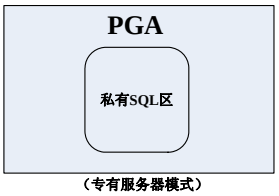
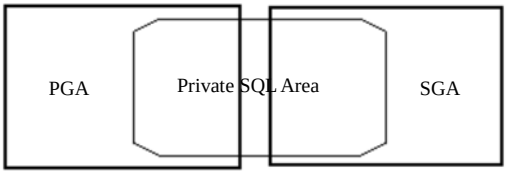


图 9-5　私有 SQL 区的位置

如果请求的是共享服务器模式（Shared Server Mode）的连接 SGA，则私有 SQL 区（Private SQL Area）的一部分位于 SGA 中，另一部分位于 PGA 中（如图 9-6 所示）。



（共享服务器模式）

图 9-6　私有 SQL 的位置

9.1.3　软件代码区（Software Code Areas）

软件代码区用于存放正在运行或可以被运行的代码。Oracle 数据库代码就存放在该区域。软件代码区的位置不同于用户程序存放的区域。软件代码区是固定的，只有在软件被更新或者重新安装的时候，软件代码区的大小才会发生改变。

9.2　Oracle 内存管理方法

早些时候，DBA 需要手工管理 Oracle 的内存，如手动调整数据库高速缓冲区（Database Buffer Cache）、重做日志缓冲区（Redo Log Buffer）、共享池（Shared Pool）、大池（Large Pool）、 Java 池（Java Pool）、流池（Streams Pool）等。手工管理 Oracle 的内存是 DBA 一项重要而又烦琐的工作！为了减轻 DBA 的工作，Oracle 引入了**自动内存管理（Automatic Memory Management）**。从 Oracle 11g 开始，Oracle 可以自动管理 SGA 和 PGA。DBA 只需指定总共的内存大小（MEMORY_TARGET），Oracle 就能够把内存在 SGA 和 PGA 之间自动进行分配。当然，Oracle 也自动对组成 SGA 的各个内存区域（如数据库高速缓冲区、重做日志缓冲区、共享池等）进行自动调整。当其中一个内存区域出现问题的时候，Oracle 能够自动对该内存区域进行调整。

注：如果初始化参数 LOCK_SGA 的值是 TRUE，则不能启动自动内存管理。

除了自动内存管理（Automatic Memory Management）之外，我们也可以启用自动 SGA 管理（Automatic Shared Memory Management）及自动 PGA 管理（Automatic PGA Memory Management）。“**自动内存管理**”使整个数据库的内存（包括 SGA、PGA）管理自动化；“自动 SGA 管理”只使 SGA 的管理自动化；“自动 PGA 管理”只使 PGA 的管理自动化，它们的区别如图 9-7 所示。

在进行内存管理之前，我们必须清楚几个和内存相关的初始化参数：

- MEMORY_TARGET，从操作系统的角度，指定 Oracle 所能使用的最大内存。它是动态参数，其值不能大于 MEMORY_MAX_TARGET。
- MEMORY_MAX_TARGET， 数据库内存的最大值。
- SGA_TARGET，SGA 的大小。

- SGA_MAX_SIZE，控制数据库启动和运行时，SGA 可以达到的最大值，该参数不能被动态改变。
- PGA_AGGREGATE_TARGET，PGA 的大小。

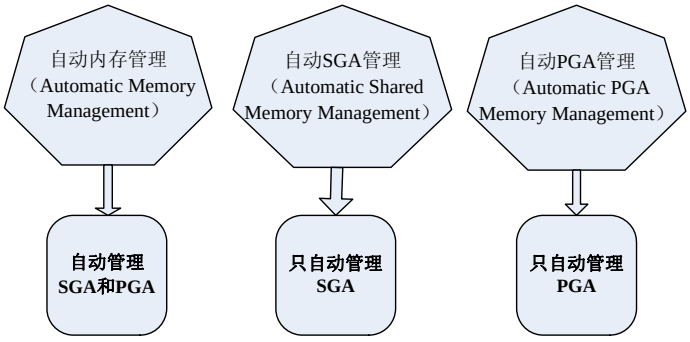


图 9-7　内存管理的几种方式

9.2.1　启用自动内存管理（Automatic Memory Management）

启用自动内存管理（Automatic Memory Management）的过程如下。特别强调的是，我们这里所说的“自动内存管理”既包括“SGA 的自动管理”，也包括“PGA 的自动管理”，它针对整个数据库内存的管理。推荐使用这种方式管理 Oracle 的内存。

STEP 01 以 SYADBA 的身份登录数据库。

STEP 02 计算出 MEMORY_TARGET 的最小值。

ⓐ 得到 SGA_TARGET 和 PGA_AGGREGATE_TARGET 的值。

```
SQL> SHOW PARAMETER TARGET
```

SQL 语句的执行结果如下所示：

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------
archive_lag_target                   integer     0
db_flashback_retention_target        integer     1440
fast_start_io_target                 integer     0
fast_start_mttr_target               integer     0
memory_max_target                    big integer 0
memory_target                        big integer 0
pga_aggregate_target                 big integer 104M
sga_target                           big integer 304M
SQL>
```

SGA_TARGET 的值是 304。

PGA_AGGREGATE_TARGET 的值是 104。

ⓑ 得到实例启动以后分配给 PGA 的最大值。

```
SQL> SELECT (value/1024/1024)||'M' FROM v$pgastat WHERE name='maximum
PGA alloca
ted';
```

SQL 语句的执行结果如下所示：

```
(VALUE/1024/1024)||'M'
----------------------------------------
82.7607421875M
SQL>
```

分配给 PGA 的最大值是 82MB。

© 计算出 MEMORY_TARGET 的最小值。

根据公式：

```
MEMORY_TARGET = SGA_TARGET + MAX(PGA_AGGREGATE_TARGET, maximum PGA
allocated)
```

计算出 MEMORY_TARGET 的最小值：

```
MEMORY_TARGET=304+MAX(104,82)=408M
```

因此 MEMORY_TARGET 的值至少是 408MB。

STEP 03 决定 MEMORY_TARGET 的值。

如果物理内存充足，MEMORY_TARGET 的值可以设置得大一些；如果物理内存不足，MEMORY_TARGET 的值可以设置得小一些，但是不能小于计算出的值（408MB），本例决定 MEMORY_TARGET 的值是 410MB。

STEP 04 决定内存的最大值。

MEMORY_MAX_TARGET 决定分配给数据库的内存的最大值，这个最大值必须是 SGA 和 PGA 的总和。MEMORY_MAX_TARGET 必须大于或者等于 MEMORY_TARGET，本例决定 MEMORY_MAX_TARGET 的值是 500MB。

查看 MEMORY_MAX_TARGET 的当前值：

```
SQL> SHOW PARAMETER MEMORY_MAX_TARGET
```

SQL 语句的执行结果如下所示：

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- -------
memory_max_target                      big integer 0
```

MEMORY_MAX_TARGET 的当前值是 0。

STEP 05 设置初始化参数。

如果数据库使用服务器参数文件（SPFILE）启动，则执行下面的 SQL 语句：

```
ALTER SYSTEM SET MEMORY_MAX_TARGET = 500M SCOPE = SPFILE;
ALTER SYSTEM SET MEMORY_TARGET = 410M SCOPE = SPFILE;

ALTER SYSTEM SET SGA_TARGET = 0 SCOPE = SPFILE;
```

```
ALTER SYSTEM SET PGA_AGGREGATE_TARGET = 0 SCOPE = SPFILE;
```

如果数据库 PFILE 启动，则编辑 PFILE，使其中几个参数的值变成如下所示：

```
MEMORY_MAX_TARGET = 500M
MEMORY_TARGET = 410M
SGA_TARGET = 0
PGA_AGGREGATE_TARGET = 0
```



重启数据库。

如果数据库使用 SPFILE，则直接启动数据库；如果使用 PFILE 启动数据库，则启动时，必须指定 PFILE 文件的名字及其路径。有关如何使用 SPFILE 及 PFILE 启动数据库，**请参考本书 6.1.3 节的内容**。



9.2.2 启用自动 SGA 管理（Automatic Shared Memory Management）

有的时候，我们只想自动管理 SGA，SGA 以外的其他内存组件想通过手工管理来实现。自动 SGA 管理是 Oracle 10g 就已经具有的功能。启动“自动 SGA 管理”的过程说明如下。



把数据库的统计级别设置成 TYPICAL 或者 ALL。

```
SQL> ALTER SYSTEM SET STATISTICS_LEVEL=TYPICAL;
System altered.
SQL>
```

其中，TYPICAL 表示收集主要的统计信息；ALL 表示收集所有的统计信息（包括操作系统统计信息）。

如果使用 PFILE 启动数据库，需要手工编辑 PFILE 中的 STATISTICS_LEVEL 参数。



通过计算，得到 SGA 目标值（SGA_TARGET）。

```
SQL> SELECT  SGA_TARGET/1024/1024||'M'
  2  FROM(
  3  SELECT (
  4    (SELECT  SUM(value)  FROM  V$SGA)  -(SELECT  CURRENT_SIZE  FROM
V$SGA_DYNAMIC_FRE
E_MEMORY)
  5  ) "SGA_TARGET"
  6  FROM DUAL
  7  );
```

SQL 语句的执行结果如下所示：

```
SGA_TARGET/1024/1024||'M'
----------------------------------------
306.875M
```

查询结果显示，SGA_TARGET 的值是 306MB。

STEP 03 设置 SGA 目标值。

```
SQL> ALTER SYSTEM SET SGA_TARGET = 310M SCOPE=BOTH;
System altered.
```

其中，SGA_TARGET 表示用于 SGA 自动调整的内存大小。SGA_MAX_SIZE 设置了 SGA 的极限，因此 SGA_TARGET 的值不能大于 SGA_MAX_SIZE 的值。本例中把 SGA_TARGET 的值设置成 310MB（比从 STEP 02 得到的值 306MB 大）。

如果使用 PFILE 启动数据库，需要手工编辑 PFILE 中的 SGA_TARGET 参数。

STEP 04 把 SGA 的相关内存组件的大小设置成 0。

```
ALTER SYSTEM SET SHARED_POOL_SIZE = 0 SCOPE=BOTH;
ALTER SYSTEM SET LARGE_POOL_SIZE = 0 SCOPE=BOTH;
ALTER SYSTEM SET JAVA_POOL_SIZE = 0 SCOPE=BOTH;
ALTER SYSTEM SET DB_CACHE_SIZE = 0 SCOPE=BOTH;
ALTER SYSTEM SET STREAMS_POOL_SIZE = 0 SCOPE=BOTH;
```

这些缓冲区不再需要 DBA 的手动调整，Oracle 会自动调整这些缓冲区的大小。

如果使用 PFILE 启动数据库，需要手工编辑 PFILE 中的这些参数。

9.2.3 启用自动 PGA 管理（Automatic PGA Memory Management）

在内存调整中，PGA 较难调整。自动 PGA 管理使 Oracle 自动管理 PGA 的各个组件，简化 DBA 的工作。如果启用“自动 PGA 管理”，则这些参数（*_AREA_SIZE）的值将被忽略。把初始化参数 PGA_AGGREGATE_TARGET 设置成非 0 值，便启用了“自动 PGA 管理”。启用“自动 PGA 管理”的过程说明如下。

STEP 01 决定 PGA_AGGREGATE_TARGET 的值。

PGA_AGGREGATE_TARGET 通常叫做 PGA 的目标值，用于对 PGA 的大小进行限定。PGA_AGGREGATE_TARGET 的值不固定，可以根据系统的需求动态改变。但是，我们可以赋予 PGA_AGGREGATE_TARGET 一个初始值。PGA_AGGREGATE_TARGET 的值通常是 SGA 的 20%。

STEP 02 修改 PGA_AGGREGATE_TARGET 的值。

```
SQL> ALTER SYSTEM SET PGA_AGGREGATE_TARGET=20M SCOPE=BOTH;
系统已更改。
```

如果数据库使用 PFILE，需要在 PFILE 中修改 PGA_AGGREGATE_TARGET，然后使用新修改的 PFILE 重新启动数据库。

当然 PGA_AGGREGATE_TARGET 的值不能设置过大，否则会提示下面的错误：

```
ORA-02097: 无法修改参数，因为指定的值无效
```

```
ORA-00840: 无法将 PGA_AGGREGATE_TARGET 修改为指定的值
```

STEP 03 把 WORKAREA_SIZE_POLICY 的值设置成 AUTO。

```
SQL> alter system set WORKAREA_SIZE_POLICY=AUTO;
系统已更改。
```

如果数据库使用 PFILE，需要在 PFILE 中修改 WORKAREA_SIZE_POLICY，然后使用新修改的 PFILE 重新启动数据库。

9.2.4 SGA 手动管理

如果 DBA 想控制 SGA 的各个内存组件，可以启用“SGA 手动管理”。启用“SGA 手动管理”的过程说明如下。

STEP 01 修改初始化参数 MEMORY_TARGET 值为 0。

```
SQL> ALTER SYSTEM SET MEMORY_TARGET = 0 SCOPE = BOTH;
系统已更改。
```

如果数据库使用 PFILE，需要在 PFILE 中修改 MEMORY_TARGET 的值，然后使用新修改的 PFILE 重新启动数据库。有关如何使用 SPFILE、PFILE 启动数据库，**请参考本书 6.1.3 节的内容**。

STEP 02 修改初始化参数 SGA_TARGET 值为 0。

```
SQL> ALTER SYSTEM SET SGA_TARGET =0 SCOPE = BOTH;
系统已更改。
```



注：如果数据库使用 PFILE，需要在 PFILE 中修改 SGA_TARGET，然后使用新修改的 PFILE 重新启动数据库。有关如何使用 SPFILE、PFILE 启动数据库，请参考本书 6.1.3 节的内容。

STEP 03 手工设置 SGA 各个内存组件的大小。

DBA 需要手工设置数据库高速缓冲区（Database Buffer Cache）、重做日志缓冲区（Redo Log Buffer）、共享池（Shared Pool）、大池（Large Pool）、Java 池（Java Pool）、流池（Streams Pool）等的大小。各个组件的控制参数如下所示：

- JAVA_POOL_SIZE 控制 Java 池的大小
- SHARED_POOL_SIZE 控制共享池的大小
- LARGE_POOL_SIZE 控制大池的大小
- DB_*_CACHE_SIZE 共有 8 个 CACHE_SIZE 参数，控制各个可用缓冲区的大小
- LOG_BUFFER 控制重做日志缓冲区的大小
- STREAMS_POOL_SIZE 控制流池的大小
- RESULT_CACHE_MAX_SIZE 控制结果缓冲区的大小
- LOCK_SGA 表示把 SGA 锁定在物理内存里

➢ …

可以使用类似于 ALTER SYSTEM SET SHARED_POOL_SIZE=*n*M SCOPE=BOTH 的命令修改上面的参数。

9.2.5 PGA 手动管理

要想禁用 PGA 自动管理，只需设置 WORKAREA_SIZE_POLICY 的值为 MANUAL 即可。如果系统运行在 PGA 手动管理模式下，DBA 需要手动设置以下的初始化参数：SORT_AREA_SIZE、HASH_AREA_SIZE、BITMAP_MERGE_AREA_SIZE、CREATE_BITMAP_AREA_SIZE。在“自动 PGA 管理”模式下，Oracle 自动管理这些参数，不需要 DBA 的干预！不过，PGA 难于调整，建议大家把 PGA 设置成自动管理。

9.3 内存信息查询

本节介绍，如果查看 Oracle 内存的使用情况。我们可以使用命令行查看内存的使用情况，也可通过图形界面工具 OEM（企业管理器）查看内存的使用情况。

9.3.1 从命令行查看 Oracle 的内存信息

可以从命令行直接查看 Oracle 内存的信息。直接从命令行返回内存信息比较快，因为每次进入企业管理器（OEM）的界面，都比较浪费时间。

1．显示 SGA 的情况

SGA 叫系统全局区，是 Oracle 中可以共享的内存。通过 SHOW 命令或者查看视图 V$SGA 可获知 SGA 的信息。

■ 使用 SHOW 命令显示 SGA 的信息

```
SQL> SHOW SGA
```

SQL 语句的执行结果如下所示：

```
Total System Global Area  523108352 bytes
Fixed Size                 1334292 bytes
Variable Size            402654188 bytes
Database Buffers         113246208 bytes
Redo Buffers               5873664 bytes
```

■ 从视图 V$SGA 获知 SGA 的信息

```
SQL>  SELECT * FROM V$SGA ;
```

SQL 语句的执行结果如下所示：

```
NAME                    VALUE
-------------------- ----------
Fixed Size             1334292
Variable Size        402654188
Database Buffers     113246208
Redo Buffers           5873664
```

- Fixed Size 是固定区域，固定区域用于存储 SGA 各组件的信息。固定区域的大小随平台和版本而变化，但是，如果是相同平台相同版本的 Oracle，则固定区域的大小是固定的、相同的。不能通过任何方式（如修改初始化参数）修改固定区域的大小。
- Variable Size 是可变区域，可变区域包括共享池、Java 池、大池。可变区域的大小可被管理员改变。
- Database Buffers 是数据库高速缓冲区。
- Redo Log Buffers 是重做日志缓冲区。

2．显示数据库高速缓冲区（Database Buffer Cache）的实际大小

有两种方法可以知道数据库高速缓冲区的实际大小：一种方法是查询视图 V$SGAINFO；另外一种方法是查询视图 V$MEMORY_DYNAMIC_COMPONENTS。

■ 查询视图 V$SGAINFO

```
SQL> SELECT NAME,BYTES/1024/1024||'M' FROM V$SGAINFO
  2  WHERE  Name='Buffer Cache Size';
```

SQL 语句的执行结果如下所示：

```
NAME                          BYTES/1024/1024||'M'
------------------------------ ------------------------------------
Buffer Cache Size               80M
```

■ 查询视图 V$MEMORY_DYNAMIC_COMPONENTS

```
SQL> SELECT SUM(current_size/1024/1024)||'M' FROM
  2  V$MEMORY_DYNAMIC_COMPONENTS
  3  WHERE component IN ('DEFAULT buffer cache','KEEP buffer cache','RECYCLE
bu
fer cache','DEFAULT 2K buffer cache','DEFAULT 4K buffer cache','DEFAULT 8K
buff
r cache','DEFAULT 16K buffer cache','DEFAULT 32K buffer cache');
```

SQL 语句的执行结果如下所示：

```
SUM(CURRENT_SIZE/1024/1024)||'M'
----------------------------------------
80M
```

从两个查询的输出结果可以看出，数据库高速缓冲区的实际大小是 80MB。

3．显示重做日志缓冲区（Redo Log Buffer）的实际大小

```
SQL> SELECT * FROM V$SGAINFO;
```

SQL 语句的执行结果如下所示：

```
NAME                             BYTES  RES
-------------------------------- ---------- ---
Fixed SGA Size                    1334292    No
Redo Buffers                      5873664    No
Buffer Cache Size               113246208   Yes
…
```

从查询结果可以看出日志缓冲区的大小是 **5873664** 字节。

4．显示共享池（Shared Pool）的实际大小

以下两个 SQL 语句都可以显示共享池的实际大小。

📖 SQL 语句 1

```
SQL> SELECT component,(current_size/1024/1024)||'M' FROM
  2  V$MEMORY_DYNAMIC_COMPONENTS
  3  WHERE component='shared pool'
  4  /
```

SQL 语句的执行结果如下所示：

```
COMPONENT       (CURRENT_SIZE/1024/1024)||'M'
----------------------------------------
shared pool         180M
```

📖 SQL 语句 2

```
SQL> SELECT NAME,BYTES/1024/1024||'M' FROM V$SGAINFO
  2  WHERE  Name='Shared Pool Size';
```

SQL 语句的执行结果如下所示：

```
NAME                          BYTES/1024/1024||'M'
------------------------------ ----------------------------
Shared Pool Size                180M
```

从以上两个 SQL 语句的结果可以看出，共享池的实际大小是 180MB。

5．显示 Java 池（Java Pool）的实际大小

以下两个 SQL 语句都可以用于显示 Java 池的实际大小。

📖 SQL 语句 1

```
SQL> SELECT component,(current_size/1024/1024)||'M' FROM
  2  V$MEMORY_DYNAMIC_COMPONENTS
  3  WHERE component='java pool';
```

SQL 语句的执行结果如下所示：

```
COMPONENT   (CURRENT_SIZE/1024/1024)||'M'
----------------------------------------
Java pool      12M
```

📖 SQL 语句 2

```
SQL> SELECT NAME,BYTES/1024/1024||'M' FROM V$SGAINFO
  2  WHERE  Name='Java Pool Size';
```

SQL 语句的执行结果如下所示：

```
NAME                          BYTES/1024/1024||'M'
------------------------------ ------------------------------------
Java Pool Size                 12M
```

本例中，Java 池的大小是 12MB。

6．显示大池的实际大小（Large Pool）

同理，我们也可以使用类似的 SQL 语句查询出大池的实际大小。

📖 SQL 语句 1

```
SQL> SELECT component,(current_size/1024/1024)||'M' FROM
  2  V$MEMORY_DYNAMIC_COMPONENTS
  3  WHERE component='large pool';
```

SQL 语句的执行结果如下所示：

```
COMPONENT              (CURRENT_SIZE/1024/1024)||'M'
----------------------------------------
large pool         16M
```

📖 SQL 语句 2

```
SQL> SELECT NAME,BYTES/1024/1024||'M' FROM V$SGAINFO
  2  WHERE  Name='Large Pool Size';
```

SQL 语句的执行结果如下所示：

```
NAME                          BYTES/1024/1024||'M'
------------------------------ ------------------------------------
Large Pool Size                16M
```

两条 SQL 语句都显示，大池的实际大小是 16MB。

7．得到共享池（Shared Pool）的剩余空间

查询视图 V$SGASTAT，可以知道共享池的剩余空间。

```
SQL> SELECT * FROM V$SGASTAT
  2  WHERE NAME = 'free memory'
  3  AND POOL = 'shared pool';
```

SQL 语句的执行结果如下所示：

```
POOL        NAME                           BYTES
----------- -------------------------- ----------
shared pool  free memory                19059452
```

输出显示，共享池的剩余空间是 19059452 字节。

8．得到大池（Large Pool）的剩余空间

同理，查询 V$SGASTAT，可以知道大池的剩余空间。

```
SQL> SELECT * FROM V$SGASTAT
  2  WHERE NAME = 'free memory'
  3  AND POOL = 'large pool';
```

SQL 语句的执行结果如下所示：

```
POOL        NAME                           BYTES
----------- -------------------------- ----------
large pool   free memory                18454128
```

本例中大池的剩余空间是 18454128 字节。

9．得到 Java 池（Java Pool）的剩余空间

从视图 V$SGASTAT 中，也可以查出 Java 池的剩余空间。

```
SQL> SELECT * FROM V$SGASTAT
  2  WHERE NAME = 'free memory'
  3  AND POOL = 'java pool';
```

SQL 语句的执行结果如下所示：

```
POOL        NAME                           BYTES
----------- -------------------------- ----------
java pool    free memory                 3063360
SQL>
```

本例中，Java 池的剩余空间是 3063360 字节。

9.3.2 用 Oracle 企业管理器查看 Oracle 内存的分配情况

Oracle 提供了命令行，用于查看 Oracle 的内存信息。如果大家不习惯用命令行，也可以借助图形界面查看 Oracle 的内存信息。图形界面简单、直观，让 Oracle 的内存信息尽收眼底。通过图形界面查看 Oracle 内存信息的步骤说明如下。

STEP 01 使用企业管理器以 SYSDBA 的身份登录数据库。

登录成功后，出现企业管理器的主界面。选择“服务器”，在“数据库配置”之下，单击“内存指导”，将出现如图 9-8 所示的界面，该界面显示了 Oracle 的内存信息。

STEP 02 显示内存信息。



图 9-8　内存管理方式

该界面有一部分显示了内存的管理方式、总共的内存大小、最大内存大小。另外一部分（如图 9-9 所示）还显示了 SGA 各个部分当前的内存分配情况。单击“内存指导”左下角的 PGA 按钮，将会显示 PGA 的信息（如图 9-10 所示）。

SGA 总大小 (MB) **308**

SGA 组件	当前分配的大小 (MB)
共享池	180
缓冲区高速缓存	92
大型池	20
Java 池	12
其他	4

图 9-9　SGA 的当前分配情况

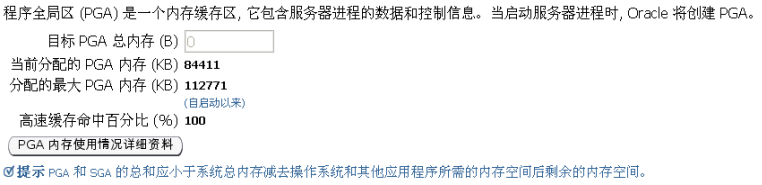


图 9-10　PGA 的使用情况



本章总结

本章对 Oracle 的内存信息进行了详细的介绍。以前，内存管理是 DBA 的重要工作之一，但如今，Oracle 使内存的管理趋于自动化。如果是在项目设计阶段，我们需要认真考虑内存对整个项目的影响，并应该从长远考虑对内存的需求；如果系统一旦建设成功，可能应该把更多的精力放在应用程序的调优之上。

第 10 章　Oracle 的发动机——进程

后台进程是 Oracle 实例的组成部分，我们已经介绍过了 Oracle 的内存结构，本章将重点介绍 Oracle 的进程结构。



10.1　进程简介

前面的章节，我们介绍了实例的概念。实例由**内存**和一系列**后台进程**组成。在讲述**后台进程**之前，我们必须清楚何为进程。

什么是进程

进程是操作系统的一种机制，它执行一系列的步骤，完成指定的任务（有些操作系统中叫作业或任务，即 Job 或 Task）。一个进程通常有它自己的私有内存区（Private Memory Area）。进程和程序的区别是，进程是动态的、有生命的，程序是静态的。程序是一系列指令的集合，进程强调的是执行过程。

数据库系统分为单进程系统（Single-Process System）和多进程系统（Multiple-Process System）。

- 单进程系统（Single-Process System）由一个用户进程完成所有的任务。
- 多进程系统（Multiple-Process System）由多个进程协同工作，每个进程负责数据库的一部分工作。多进程系统能提高整个数据库系统的性能。当前主流的数据库都是多进程系统。

进程的分类

进程分为两类：用户进程（User Process）与 Oracle 进程，如图 10-1 所示。当用户运行应用程序或者使用 Oracle 客户端（如 SQL*Plus）的时候，创建用户进程。Oracle 进程运行在 Oracle 数据库服务器端。Oracle 进程分为服务器进程（Server Process）和后台进程（Background Process）。服务器进程用于处理连接到数据库的用户进程的请求。服务器进程分为专有服务器进程（Dedicated Server Process）和共享服务器进程（Shared Server Process），它们的功能相同，唯一不同的是专有服务器进程只为特定的用户进程服务，如图 10-2 所示，共享服务器进程为所有的用户进程服务。

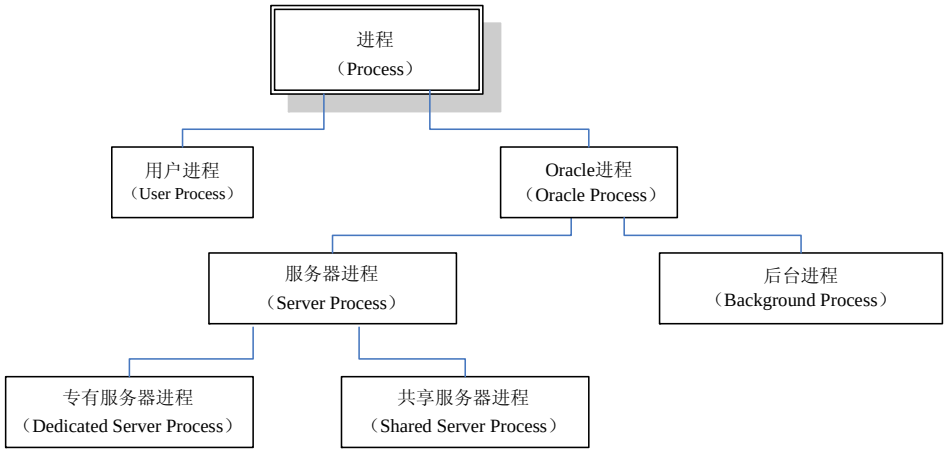


图 10-1　进程分类

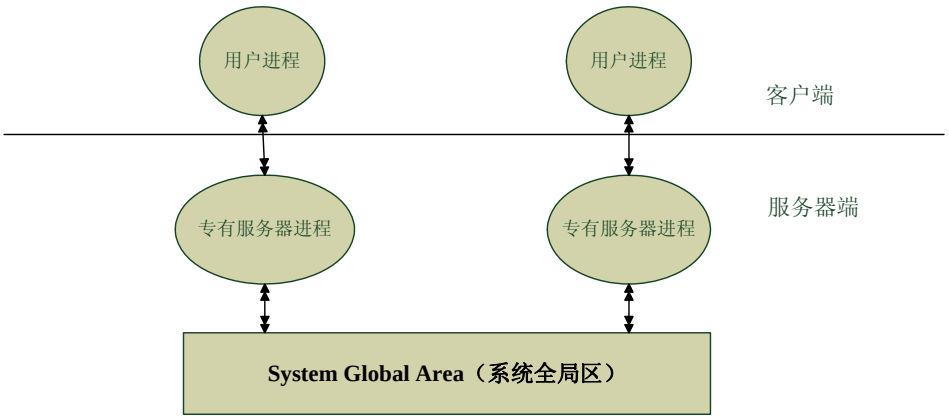


图 10-2　专有服务器进程的功能

用户进程（User Process）

当用户运行一个应用程序（如 Pro*C）或者 Oracle 工具（如 SQL*Plus）的时候，Oracle 会创建一个用户进程。

和用户进程比较相似的有两个概念：连接（Connection）与会话（Session）。

- 连接（Connection）是用户进程到数据库实例之间的一条通信路径。这条路径使用不同的通信机制。如果用户进程和实例在同一台机器上，则使用进程间的通信机制（IPC）；如果用户进程和实例在不同的机器上，则通过网络进行通信。
- 会话（Session）是一个用户到数据库的一次特殊的连接。启动 SQL*Plus 时，必须提供用户名和密码，然后建立一个会话。会话是有生命的，会话的生命从用户登录开始，到用户退出 Oracle 而结束，这就是会话的生命周期。一个用户没有登录就是一个用户，一旦用户登录数据库以后就变成了会话。一个用户可以创建多个会话。如用 100 个客户端使用相同的用户名 tu 同时登录一个数据库，就创建了 100 个会话。用户和会话的关系如图 10-3 所示。

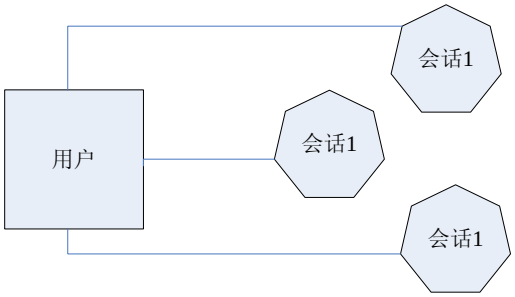


图 10-3　用户和会话的关系

会话的两种状态

会话有两种状态：活动状态（ACTIVE）、非活动状态（INACTIVE）。会话的状态取决于会话是否正在做 SQL 调用。如果一个会话正在做 SQL 调用，则该会话的状态为“活动”；如果该会话未做 SQL 调用，则该会话的状态为“非活动”。

服务器进程（Server Process）

Oracle 创建服务器进程，并用服务器进程来处理用户进程的请求。在某些情况下，如果应用程序和 Oracle 在同一台机器上，为了减少系统的负载，Oracle 会把用户进程和服务器进程合并成一个进程。如果应用程序和 Oracle 在不同的机器上，则用户进程通过网络和服务器进程通信。

服务器进程负责下面的工作：

- 对 SQL 语句进行语法解析和执行。
- 需要数据的时候，如果数据不在 SGA 中，服务器进程会把数据从磁盘上的数据文件中读取到 SGA 的数据高速缓冲区中（Database Buffers）。
- 把结果返回给应用程序。

不同的进程位置不同，有的处在客户端，有的驻留在数据库服务器端，进程的位置如图 10-4 所示。

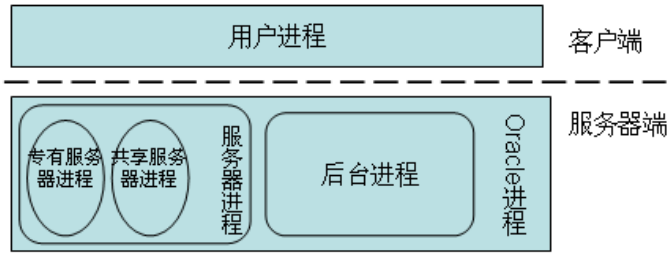


图 10-4　进程的位置

从图 10-4 中可以看出，除了用户进程位于客户端外，其他进程都驻留在数据库服务器端。

10.2 Oracle 后台进程（Background Process）

为了最大化性能和容纳更多的用户，一个多进程系统会使用额外的 Oracle 进程，这些进程就是后台进程。一个实例可以有多个后台进程。但是，并不是每一个后台进程都会出现。通过视图 V$BGPROCESS 可以查看后台进程的信息。Oracle 有下面的后台进程：

- 归档进程（Archiver Processes，ARCn）
- 校验点进程（Checkpoint Process，CKPT）
- 数据库书写进程（Database Writer Process，DBWn）
- 作业队列进程（Job Queue Process）
- 日志书写进程（Log Writer Process，LGWR）
- 进程监控进程（Process Monitor Process，PMON）
- 队列监控进程（Queue Monitor Process，QMNn）
- 恢复进程（Recover Process，RECO）
- 系统监控进程（System Monitor Process，SMON）
- 其他后台进程（Other Background Process）

启动实例时，Oracle 会自动启动后台进程。但是，并不是所有的后台进程都启动。启动后台进程是实例启动的一部分。数据库启动完成后，可从警报文件中查看 Oracle 到底启动了哪些后台进程。

1．归档进程（Archive Processes，ARCn）

当发生一次日志切换（Log Switch）时，归档进程（ARCn）将把重做日志文件（Online Redo Log Files）复制到指定的存储设备上（如图 10-5 所示）。归档进程只有数据库运行在归档模式（ARCHIVELOG Mode）且启用自动归档（Automatic Archiving）的情况下才会出现。如果数据库运行在非归档模式（NOARCHIVELOG Mode），Oracle 不需要归档进程（ARCn）。



图 10-5　归档进程的功能

一个实例可以有 10 个归档进程（ARC0 到 ARC9）。需要的时候，日志书写进程会自动启动归档进程。DBA 不能控制归档进程的数量，也不能启动和关闭归档进程。归档进程的启动由日志书写进程完成。虽然我们不能控制正在运行的归档进程的数量，但是，我们能够通过初始化参数 ARCHIVE_MAX_PROCESSES 控制可以启动的归档进程的最大数量。使用 ALTER SYSTEM 命令可以动态修改 ARCHIVE_MAX_PROCESSES 的值。

2．校验点进程（Checkpoint Process，CKPT）

当一个校验点（Checkpoint）发生时，Oracle 必须更新所有数据文件的头部，以记录校验点的详细信息。这项工作由校验点进程（CKPT）完成。校验点进程并不会写数据块（Blocks），这项工作总由数据库书写进程（DBWn）完成。

3．数据库书写进程（Database Writer Process，DBWn）

数据库书写进程（DBWn）是后台进程，它负责把数据库高速缓冲区中的数据写到磁盘上的数据文件中（如图 10-6 所示）。当数据库高速缓冲区（Database Buffer Cache）中的一个缓冲器（Buffer）被修改，这个缓冲器被标识为“脏（Dirty）”，这样的缓冲器称为脏缓冲器（Dirty Buffer），被修改的数据叫做脏数据（Dirty Data）。利用 Oracle 的最近最少使用机制（Least Recently Used，LRU），如果一个缓冲器最近最少使用，Oracle 把这样的缓冲区标识为“冷（Cold）”，我们把这样的缓冲区叫“冷缓冲器（Cold Buffer）”。数据库书写进程的主要任务是将“脏的”且“冷的”缓冲器写入磁盘，使这些缓冲器变得“干净（Clean）”。“干净”的缓冲器可以容纳从磁盘上读来的数据。Oracle 运行的时候，缓冲区中不断被填入数据，有些数据会被用户弄脏，未用缓冲器的数目逐渐减少，当未用的缓冲器下降到很少很少，以至于服务器进程想要从磁盘读入数据块到内存中，而无法找到未用的缓冲器时，数据库书写进程将把最近最少使用的脏缓冲器中的数据写入磁盘。下面的情况发生时，Oracle 会通知数据库书写进程写脏数据到磁盘：

- 当 Oracle 扫描缓冲区的一个临界值，如果不能发现可用的缓冲器，Oracle 会通知数据库书写进程把脏数据（被修改过的数据）写到磁盘。
- 当发生检查点（Checkpoint）时，LGWR 将通知数据库书写进程把脏数据写入磁盘。

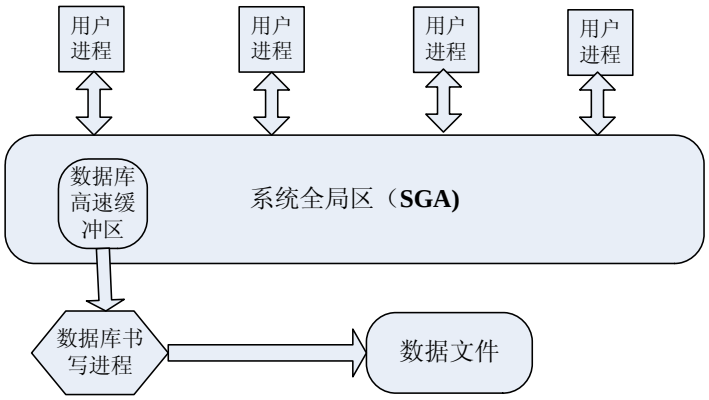


图 10-6　数据库书写进程的工作原理

通常，一个数据库书写进程能够满足需求。为了提高数据的性能，我们可以配置多个数据库书写进程同时写数据文件。这些数据库书写进程可以从 DBW1 到 DBW9 或从 DBWa 到 DBWj。数据库书写进程的数量由初始化参数 DB_WRITER_PROCESSES 控制。初始化参数 DB_WRITER_PROCESSES 的设置是基于 CPU 的数量的，如果系统只有一个 CPU，即使设置了多个数据库书写进程，也只有一个数据库书写进程有效，在这种情况下，设置

多个数据库书写进程并不会提高数据库的性能。

4．作业队列进程（Job Queue Process）

作业队列进程用于批处理（Batch Processing），它主要负责作业（Jobs）的运行。用户可以定时作业。用户可以定义作业的开始时间以及执行间隔。间隔时间到了的时候，作业队列进程会执行指定的作业。

作业队列的工作原理如图 10-7 所示。队列协调进程（Coordinator Process）CJQ0 从系统表 JOB$中得到要被执行的作业，然后分裂一个子进程（Slave Processes），该子进程真正执行选定的作业。

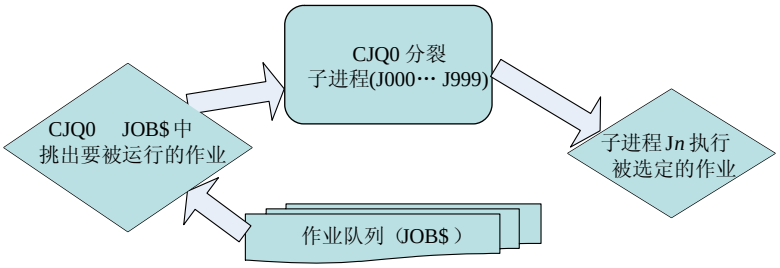


图 10-7　作业队列的工作原理

初始化参数 JOB_QUEUE_PROCESSES 用于设置一个实例中作业队列进程的最大数量。包 DBMS_JOB 用于作业的管理。JOB_QUEUE_PROCESSES 的默认值是 1000。如果 JOB_QUEUE_PROCESSES 的值为 0，则队列协调进程（Coordinator Process）不会被启动，也就是作业队列的功能被取消。作业队列进程是可选的后台进程，即这些进程的失败不会引起实例失败。

5．日志书写进程（Log Writer Process，LGWR）

日志书写进程负责重做日志缓冲区（Redo Log Buffer）的管理，它把重做日志缓冲区的内容写到磁盘上的重做日志文件中（Online Redo Log Files），如图 10-8 所示。

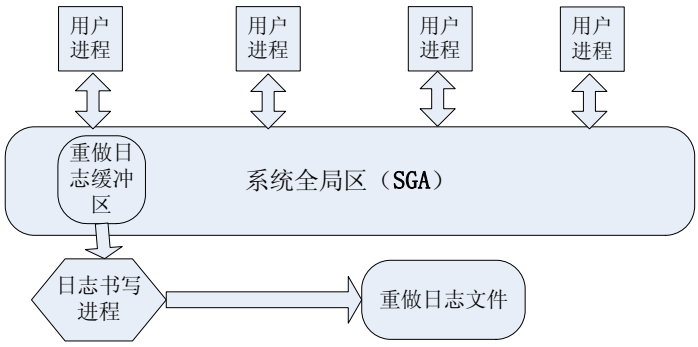


图 10-8　日志书写进程的工作原理

重做日志缓冲区（Redo Log Buffer）被循环利用。当用户输入一条 SQL 语句对数据库

进行修改的时候，服务器进程把与修改相关的信息以重做日志条目（Redo Entries）的形式复制到重做日志缓冲区中。通常，日志书写进程会以最快的速度把这些重做日志条目写到重做文件（Online Redo Log Files）中，以便在重做日志缓冲区中预留出更多的空余空间。服务器进程负责复制新的重做日志条目到日志缓冲区中。在下面的条件下，日志书写进程会把重做日志缓冲区中的重做日志条目写入磁盘：

- 当用户进程提交一个事务
- 每隔三秒钟
- 日志缓冲区的 1/3 已被填满
- 当数据库书写进程将修改的数据写入磁盘

当用户提交 COMMIT 语句时，重做日志条目被放入日志缓冲区，日志书写进程立即把这些重做日志条目从重做日志缓冲区写到重做日志文件中。但是，被修改的数据并没有被立即写到磁盘上的数据文件中，而是延迟了，这种方式叫“快速提交机制（Fast Commit Mechanism）”。

一个事务（Transaction）被提交，日志书写进程会把与它相关的重做日志条目写入重做日志文件中。有时，如果重做日志缓冲区的空间不足，日志书写进程也会把那些未提交的事务的日志条目写入重做日志文件中。当这个事务被提交后，写到重做日志文件中的重做日志条目将变成永久改变。

当一个事务提交时，被赋予一个系统修改号（System Change Number，SCN），它随事务的重做日志条目一起记录在重做日志文件中。由于 SCN 记录在日志文件中，以至于在集群（Real Application Clusters，RAC）环境或者分布式（Distributed Databases）环境下，恢复操作可以同步。

6．进程监控进程（Process Monitor Process，PMON）

进程监控进程扮演**“进程清道夫”**的角色，它负责进程的管理。当用户进程失败，进程监控进程将清除该用户进程占用的数据库高速缓冲区及其他各种资源。

进程监控进程会定期检查调度进程（Dispatcher）和服务器进程（Server Processes）的状态，并试图重新启动那些被停止的进程（这并不是 Oracle 希望停止的）。进程监控进程还会把调度进程和实例的信息注册到监听器（Listener）。进程监控进程还可以被其他进程所调用。

7．队列监控进程（Queue Monitor Process，QMNn）

队列监控进程是可选的后台进程，它负责监控消息队列（Message Queues）。一个实例中可以有 10 个队列监控进程。队列监控进程是可选的后台进程，也就是说，这种进程的失败不会引起实例的失败。

8．恢复进程（Recover Process，RECO）

恢复进程主要用于处理失败的分布式事务（Distributed Transaction）。当一个分布式事务失败，恢复进程会自动连接到和这个失败的分布式事务相关的远程数据库。有时，由于

某些原因，恢复进程不能够成功连接到远程数据库，但是，一定的时间间隔后，恢复进程又会自动重新连接远程数据库。一旦连接上远程数据库，恢复进程会自动解决有问题的事务。任何悬而未决的事务的运行将从每一个数据库的悬挂事务表（Pending Transaction Table）中清除。总之，恢复进程用于处理分布式事务。

9．系统监控进程（System Monitor Process，SMON）

数据库启动时，系统监控进程负责实例恢复。系统监控进程还负责清理不再使用的**临时段**。如果需要，其他进程还可以调用系统监控进程。在集群环境下，系统监控进程还负责 CUP 故障和实例失败的恢复。系统监控进程的职责如下：

- 实例恢复
- 清除临时段（Temporary Segments）
- 合并连续的区（Extents）
- 被其他进程所调用

10．其他后台进程（Other Background Process）

除了上面的后台进程，Oracle 还有其他后台进程：CMS、DIA0、DIAG、MNC、FBDA GTX0-j、MMAN、MMNL、MMON、PSP0、SMCO、VKTM。这里就不一一进行讲述。



10.3 进程管理与维护

本节将描述如何对进程进行管理和维护，重点是如何知道进程的运行状态及进程的数量修改。



10.3.1 查看 Oracle 的后台进程

查询视图 V$BGPROCESS，可以知道数据库中正在运行的后台进程：

```
SELECT * FROM V$BGPROCESS
WHERE PADDR != '00'
ORDER BY NAME;
```

SQL 语句的执行结果如下所示：

```
PADDR   PSERIAL#       NAME       DESCRIPTION            ERROR
29294D78       3       ARC0       Archival Process 0         0
29295850       1       ARC1       Archival Process 1         0
29296328       1       ARC2       Archival Process 2         0
29289FF8       2       CKPT       checkpoint                 0
2928C080       1       DBW0       db  writer process 0       0
2928CB58       1       LGWR       Redo etc.                  0
292869C0       1       PMON       process cleanup            0
```

```
29299960        1       QMNCAQ      Coordinator                 0
2928E108        1       RECO        distributed recovery        0
2928D630        1       SMON        System Monitor Process      0
…
```

从查询结果可以看出，数据库已经启动了归档进程（ARC0）、校验点进程（CKPT）、数据库书写进程（DBW0）等后台进程。

如果数据库运行在 UNIX 平台，执行命令 ps -ef | grep ora_也可知道哪些后台进程正在运行。



10.3.2 查看数据库中的会话

V$SESSION 用于存放和会话相关的信息。查询该视图，可以清楚数据库中有哪些会话。下面的 SQL 语句将列出数据库中的会话。

```
SQL> SELECT * FROM V$SESSION WHERE type='USER';
```

SQL 语句的执行结果如下所示：

```
SADDR       SID     SERIAL#     AUDSID  PADDR       USER#   USERNAME …
-------- --- ------- ------ -------- ----- ---------- ------ ----------
5A93DC3C    142     14          119     5A871FA8    5       SYSTEM
5A94000C    144     11          117     5A871AAC    5       SYSTEM
5A9411F4    145     14          115     5A8715B0    5       SYSTEM
5A9435C4    147     3           0       5A870BB8     0
5A9447AC    148     18          118     5A8710B4    58      TEST
5A945994    149     25          0       5A8706BC    0
6 行 已选择
```

从输出结果可以看出，系统中共有 6 个会话。



10.3.3 最热门的话题——提高数据库最大会话数

修改最大会话数的实质是修改初始化参数 PROCESSES。Oracle 的最大会话数是由初始化参数 PROCESSES 控制的，并不是由初始化参数 SESSIONS 控制的，初始化参数 SESSIONS 的值是由 PROCESSES 衍生出来的（也就是根据初始化参数 PROCESSES 计算出来的），SESSIONS 默认值等于 1.1×PROCESSES+5。修改最大会话数的方法说明如下。

STEP 01 进入初始化参数文件所在的目录。

初始化参数文件（PFILE）和服务器参数文件（SPFILE）位于以下的路径。

- ORACLE_HOME/dbs（Linux 和 UNIX 平台）
- ORACLE_HOME\database(Windows 平台)

其中，ORACLE_HOME 表示 Oracle 的主目录。

STEP 02 如果数据库使用的是 SPFILE，则根据 SPFILE 生成初始化参数文件

（PFILE），否则直接执行下一步。

```
    SQL> CREATE PFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
      2  FROM  SPFILE = 'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
      3 /
    文件已创建
```

其中，SPFILEZDB.ORA 是 SPFILE；INITZDB.ORA 是 PFILE。

STEP 03 编辑初始化参数文件（PFILE）。

编辑初始化参数文件 INITZDB.ORA，修改初始化参数 PROCESSES 的值。

STEP 04 关闭数据库。

```
    SQL> SHUTDOWN IMMEDIATE;
```

STEP 05 重建 SPFILE。

```
    SQL> CREATE SPFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
    2   FROM  PFILE  =  'C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
    3 /
    文件已创建
```

STEP 06 重启数据库。

```
   SQL> STARTUP
```

这里将用新建的 SPFILE 启动数据库。

10.3.4 终止会话（Terminating Sessions）

有些时候，需要终止会话。终止会话将会：

- 回滚该会话的活动事务（Active Transactions）
- 释放该会话占用的资源

终止会话的过程说明如下。

STEP 01 查询会话的状态。

```
    SELECT SID,SERIAL#,STATUS,SERVER
    FROM V$SESSION
    WHERE USERNAME = 'JB';
```

SQL 语句的执行结果如下所示：

```
    SID SERIAL# STATUS
    ----- --------- --------
    7   15      ACTIVE
    12  63      INACTIVE
```

Oracle 通过系统标识符（System Identifier，SID）和序列号（Serial Number，SERIAL#）唯一标识一个会话。本例列出用户 JB 的两个会话。其中，一个会话处于活动状态，另外一个处于非活动状态。

STEP 02 终止会话。

```
ALTER SYSTEM KILL SESSION '7,15';
```

本例终止第一个会话（SID=7，SERIAL#=15）。

活动会话的终止。当终止会话时，如果终止的会话正在处理事务，该事务将被立刻回滚，并会返回错误信息 ORA-00028: your session has been killed；但是，会话执行的某些操作（如网络 I/O 操作）不能被立即终止，直到操作执行完成，Oracle 会把这样的会话状态标识成“KILLED”，SERVER 的值将变成“PSEUDO”。

非活动会话的终止。当终止非活动会话后，该会话的状态也被标识成“KILLED”。这些信息会一直保留在数据字典 V$SESSION 中，直到用户再次试图使用该会话。

STEP 03 再次查询会话的状态。

```
SELECT SID, SERIAL#, STATUS, SERVER
FROM V$SESSION
WHERE USERNAME = 'JB';
```

SQL 语句的执行结果如下所示：

```
SID SERIAL# STATUS     SERVER
----- -------- --------- ---------
7   15      KILLED     PSEUDO
12  63      INACTIVE   DEDICATED
```

10.3.5 修改归档进程（ARCn）的数量

归档进程的数量是由初始化参数 LOG_ARCHIVE_MAX_PROCESSES 控制的。当归档比较滞后的时候，Oracle 会自动启动更多的归档进程，但是，启动的归档进程数量不能超过 LOG_ARCHIVE_MAX_PROCESSES 的限制，LOG_ARCHIVE_MAX_PROCESSES 只是控制归档进程的**最大数量**。实际启动多少个归档进程由 Oracle 自动决定。归档工作负载很重时，Oracle 会自动启动更多的归档进程。修改归档进程的数量有两种方法。

方法 1

执行下面的命令：

```
SQL>ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=4 SCOPE=both;
System altered.
```

这条命令把归档进程的最大数量提高到 4。

LOG_ARCHIVE_MAX_PROCESSES 是初始化参数，并且可以动态修改！



注：如果系统使用 SPFILE 启动数据库，并且初始化参数的值可以动态修改，可选择方法 1，否则选择方法 2。

方法 2

STEP 01 进入初始化参数文件所在的目录。

初始化参数文件（PFILE）和服务器参数文件（Server Parameter File）位于以下的路径：

- ORACLE_HOME/dbs（Linux 和 UNIX 平台）
- ORACLE_HOME\database（Windows 平台）

其中，ORACLE_HOME 表示 ORACLE 的主目录。

STEP 02 如果数据库使用的是 SPFILE，则根据 SPFILE 生成初始化参数文件（PFILE），否则直接执行下一步。

```
    SQL> CREATE PFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
      2  FROM  SPFILE = 'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
      3 /
    文件已创建
```

其中，SPFILEZDB.ORA 是 SPFILE；INITZDB.ORA 是 PFILE。

STEP 03 编辑初始化参数文件（PFILE）。

编辑初始化参数文件 INITZDB.ORA，修改初始化参数 LOG_ARCHIVE_MAX_PROCESSES 的值。

STEP 04 关闭数据库。

```
    SQL> SHUTDOWN IMMEDIATE;
    数据库已经关闭。
    已经卸载数据库。
    ORACLE 例程已经关闭。
```

STEP 05 重建 SPFILE。

```
    SQL> CREATE SPFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
    2   FROM  PFILE  =  'C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
    3 /
    文件已创建
```

STEP 06 重启数据库。

```
    SQL> STARTUP
    ORACLE 例程已经启动。
    Total System Global Area  426852352 bytes
```

```
Fixed Size                  1333648 bytes
Variable Size             314574448 bytes
Database Buffers          104857600 bytes
Redo Buffers                6086656 bytes
数据库装载完毕。
数据库已经打开。
```

这里将用新建的 SPFILE 启动数据库。

STEP 07 查看数据库是否已经启动了相应数量的归档进程。

```
SELECT * FROM V$BGPROCESS WHERE paddr<>'00';
```

10.3.6 修改数据库书写进程（DBWn）的数量

为了分布 I/O，减少单个数据库书写进程的负担，我们需要增加数据库书写进程的数量。对于多 CPU 系统，更应该启用多个数据库书写进程！数据库书写进程的数量由初始化参数 DB_WRITER_PROCESSES 控制。当数据库启动时，Oracle 会根据 DB_WRITER_PROCESSES 的值启动相应数量的数据库书写进程。数据库启动以后，不能够动态增加或减少数据库书写进程的数量。**修改数据库书写进程数量的方法请参考本书 10.3.5 节中的方法 2。**

10.3.7 修改作业队列进程（Job Queue Process）的数量

作业队列进程的最大数量由初始化参数 JOB_QUEUE_PROCESSES 控制。JOB_QUEUE_PROCESSES 的修改和归档进程数量的修改类似。有关归档进程数量的修改，**请参考本书 10.3.5 节的内容**。

10.3.8 查看每个进程占用的系统资源

在不同的操作系统，可以使用不同的命令查看 Oracle 进程占用的系统资源。

方法 1：Solaris 使用 prstat 命令查看每个进程消耗的系统资源

在操作系统命令行中执行下面的命令。

```
$ prstat -a
   PID   USERNAMESIZE    RSS    STATE PRI NICE TIME    CPU     PROCESS/NLWP
   1354  Oracle  4468K   4012K   cpu0    59  0   0:00:01 1.4%    prstat/1
   329   Oracle  249M    12M     run     59  0   0:00:59 0.2%    Oracle/11
   325   Oracle  251M    13M    sleep    59  0   0:00:35 0.1%    Oracle/11
   169   root    2464K   1428K   sleep   59  0   0:00:00 0.0%    inetd/1
 NPROC  USERNAMESIZE    RSS     MEMORY      TIME        CPU
    28   Oracle  5551M   346M    51%         0:11:11     2.0%
```

```
   30   root    77M     35M     5.1%        0:01:04     0.0%
    1   smmsp   4340K   1132K   0.2%        0:00:01     0.0%
    1   daemon  2488K   1376K   0.2%        0:00:00     0.0%
Total:  60 processes, 171 lwps, load averages: 0.05, 0.20, 0.21
```

其中，PID 表示进程的 ID；USERNAME 表示进程所有者使用的登录用户名；SIZE 表示进程在虚拟内存中的总长度；RSS 表示进程常驻部分的长度（K、M、G）；STATE 表示进程的状态；CPU 表示进程正在 CPU 上运行；SLEEP 表示进程正在等待一个事件的完成；RUN 表示进程正处在运行队列；ZOMBIE 表示进程终止并且没有父进程等待；STOP 表示进程是停止的；PRI 表示进程的优先权；NICE 表示在计算进程优先权中使用的值；TIME 表示进程执行的累积时间；CPU 表示进程对 CPU 的利用率；PROCESS/NLWP 表示进程名/进程中的轻量级进程号码。

方法 2：Linux 及 HP-UNIX 使用 top 命令查看每个进程的资源消耗情况

Linux 中 top 命令的输出结果如下所示：

```
09:53:54  up 4 days, 19:43,  1 user,  load average: 0.70, 1.12, 0.82
232 processes: 230 sleeping, 2 running, 0 zombie, 0 stopped
CPU states: cpu user    nice    system  irq softirq     iowait  idle
            total   2.2%    0.0%    0.9%    0.0%    0.2%    3.2%    93.3%
            cpu00   0.0%    0.0%    0.0%    0.0%    0.9%    0.0%    99.0%
            cpu01   0.9%    0.0%    0.0%    0.0%    0.0%    2.8%    96.2%
            cpu02   4.7%    0.0%    0.0%    0.0%    0.0%    11.3%   83.9%
Mem:  3611320k av, 3592052k used,   19268k free,       0k shrd,    4944k buff
                2608232k actv,  311528k in_d,   55340k in_c
Swap: 1534168k av,   82584k used, 1451584k free                 2879280k
cached
  PID    USER   PRI NI SIZE  RSS  SHARE STAT %CPU  %MEM  TIME CPU COMMAND
  20958  Oracle 16  0  113M  112M 111M  R    1.0   3.2   0:45 2   Oracle
   3474  Oracle 22  0  1308  1308 828   R    0.4   0.0   0:00 5   top
     1   root   15  0  204   172  140   S    0.0   0.0   1:29 3   init
     2   root   RT  0  0     0    0     SW   0.0   0.0   0:00 0   migration/0
```

其中，processes 表示显示自最近一次刷新以来的运行进程总数。这些进程被分为正在运行的、休眠的、停止的等很多种类；CPU states 表示显示 CPU 的统计情况，包括 CPU 的用户时间百分比、系统时间百分比、空闲时间百分比等；Mem 表示显示内存使用情况，其中包括总的内存、空闲内存、已用内存；Swap 表示交换空间统计，其中包括总的交换空间、可用交换空间、已用交换空间；PID 表示每个进程的 ID；PPID 表示每个进程的父进程 ID；UID 表示每个进程所有者的 UID；USER 表示每个进程所有者的用户名；PRI 表示每个进程的优先级别；NI 表示该进程的优先级值；SIZE 表示该进程的代码大小加上数据大小再加上堆栈空间大小的总数；RSS 表示该进程占用物理内存的总数量，单位是 KB；SHARE 表示该进程使用共享内存的数量；STAT 表示该进程的状态，其中 S 代表休眠状态；D 代表不可中断的休眠状态；R 代表运行状态；Z 代表僵死状态；T 代表停止或跟踪状态；TIME 表示该进程自启动以来所占用的总 CPU 时间；%CPU 表示该进程使用的 CPU 时间

占总时间的百分比；%MEM 表示该进程占用的物理内存占总内存的百分比；COMMAND 表示该进程的命令名称。



本章总结

本章介绍 Oracle 的进程结构，并重点介绍了 Oracle 的后台进程。对于开发人员来说，只需了解每个进程的大致功能即可。我们可以通过参数控制 Oracle 的大部分后台进程，但是，我们不能从操作系统中随意杀掉 Oracle 的后台进程，那样很可能导致数据库宕机。如果你是 DBA，最常用的知识点是如何提高最大会话数和如何查看每个进程消耗的系统资源。

第 11 章　数据库物理结构信息的仓库
——Oracle 控制文件

本章将对控制文件进行介绍。学习控制文件时，应该理解控制文件的内容及其重要性，并区别控制文件与其他 Oracle 物理文件。

11.1　控制文件（Control Files）简介

正如本章标题所示，控制文件（Control Files）其实是一个仓库，该仓库用于存放数据库物理结构信息（如图 11-1 所示）。它是一个很小的二进制文件。一个控制文件只属于一个数据库。创建数据库时，创建控制文件。当数据库的物理结构发生改变时，Oracle 会自动更新控制文件。用户不能手工编辑控制文件，控制文件的修改由 Oracle 完成。

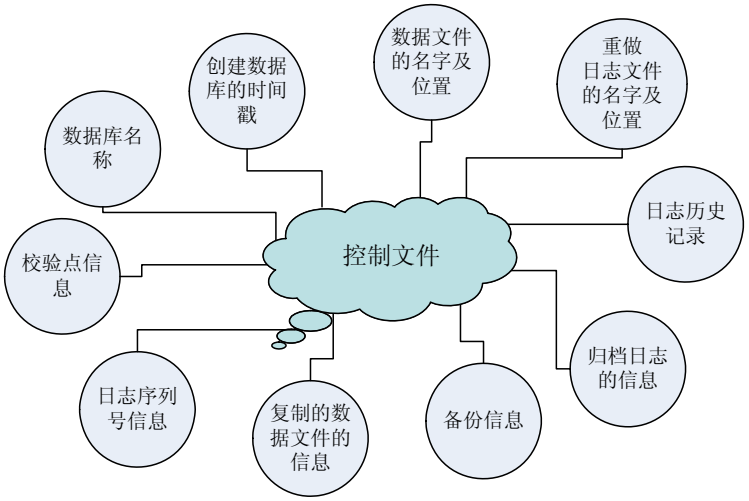


图 11-1　控制文件存放的信息

数据库的启动和正常运行都离不开控制文件。启动数据库时，Oracle 从初始化参数文件中获得控制文件的名字及位置，并打开控制文件，然后从控制文件中读取数据文件和重做日志文件（Online Redo Log Files）的信息，最后打开数据库。数据库运行时，Oracle 会

更改控制文件，所以，一旦控制文件损坏，数据库将不能正常运行。

控制文件的重要性

当数据库打开时，控制文件必须有效。如果控制文件出现错误，数据库将不能正常运行。控制文件的损坏或者丢失将会导致整个数据库的损坏，所以，必须保护好控制文件。当对控制文件进行维护时，一定要小心谨慎！

数据库物理结构改变，为什么要备份控制文件

当增加、重命名、删除一个数据文件或者一个重做日志文件时，Oracle 服务器进程（Server Process）会立即更新控制文件以反映数据库结构的这种变化。所以，Oracle 总是告诫我们，在数据库的结构发生变化后，要备份控制文件。日志写进程（LGWR）负责把当前日志序列号记录到控制文件中。校验点进程（CKPT）负责把校验点的信息记录到控制文件中。归档进程负责把归档日志的信息记录到控制文件中。

控制文件和初始化参数 CONTROL_FILES 的关系

初始化参数 CONTROL_FILES 的值记录控制文件的位置。通常，DBA 应该镜像控制文件，把每个控制文件分布到不同的物理磁盘，当发生灾难时，即使其中一个控制文件损坏，数据也不会丢失，更不至于使整个数据库陷于瘫痪。



11.2 查看数据库中的控制文件

通过查看数据字典视图 V$CONTROLFILE 或者查看初始化参数 control_files 均可知道控制文件的名字及位置。

■ 查看视图 V$CONTROLFILE

```
SELECT * FROM V$CONTROLFILE;
```

SQL 语句的执行结果如下所示：

```
STATUS  NAME                    IS_RECOVERY_DEST_FILE BLOCK_SIZE   FILE_SIZE_BLKS
        F:\NEWDB\ZDB\CONTROL01.CTL  NO          16384       596
        F:\NEWDB\ZDB\CONTROL02.CTL  NO          16384       596
        F:\NEWDB\ZDB\CONTROL03.CTL  NO          16384       596
```

■ 查看初始化参数 control_files

```
SHOW PARAMETER control_files
```

SQL 语句的执行结果如下所示：

```
NAME                          TYPE          VALUE
------------------------------------ --------
control_files                  string       F:\NEWDB\ZDB\CONTROL01.CTL,
                                            F:\NEWDB\ZDB\CONTROL02.CTL,
```

```
F:\NEWDB\ZDB\CONTROL03.CTL
```

以上两种方法均可以知道，数据库共有 3 个控制文件（F:\NEWDB\ZDB\CONTROL01.CTL、F:\NEWDB\ZDB\CONTROL02.CTL、F:\NEWDB\ZDB\CONTROL03.CTL）。



11.3 DBA 必做的事情——镜像控制文件（Multiplex Control Files）

为了更好地保护控制文件，需要对控制文件进行镜像。每个数据库至少需要两个控制文件，这些控制文件互为镜像。值得一提的是，每个控制文件应该分布在不同的物理磁盘，当其中一块磁盘损坏，丢失或者损坏的也只是一个控制文件，其他磁盘上的控制文件仍然能够保证数据库的完整性。镜像控制文件的过程说明如下（本例中数据库已有 3 个控制文件 F:\NEWDB\ZDB\CONTROL01.CTL、F:\NEWDB\ZDB\CONTROL02.CTL、F:\NEWDB\ZDB\CONTROL03.CTL）。

STEP 01 以具有 SYSDBA 权限的用户身份登录数据库。

```
SQL> CONNECT sys@myzdb as SYSDBA
Enter password:
Connected.
```

STEP 02 关闭数据库。

```
SQL> SHUTDOWN IMMEDIATE;
```

STEP 03 生成镜像文件。

使用操作系统命令作复制控制文件 CONTROL03.CTL 到新的路径 D:\NEWDB\，并重命名成 CONTROL04.CTL，严格地说，CONTROL04.CTL 应该位于另外一块独立的磁盘，本例只是实验，所以 CONTROL03.CTL、CONTROL04.CTL 可以位于同一块磁盘。

STEP 04 进入初始化参数文件所在的默认路径。

- ORACLE_HOME/dbs（Linux 和 UNIX 平台）
- ORACLE_HOME\database(Windows 平台)

其中，ORACLE_HOME 表示 Oracle 的主目录。

STEP 05 修改 PFILE（本例中是 INITZDB.ORA）。

编辑初始化参数文件 INITZDB.ORA，在参数 control_files 的后面加上 'D:\NEWDB\CONTROL04.CTL'，CONTROL04.CTL 是新的镜像文件。即：

```
*.control_files='F:\NEWDB\ZDB\CONTROL01.CTL','F:\NEWDB\ZDB\CONTROL02.CTL', ' F:\NEWDB\ZDB\CONTROL03.CTL', 'D:\NEWDB\CONTROL04.CTL'
```

为了保险起见，我们可以镜像 3 个甚至更多的控制文件。

STEP 06 根据 PFILE 生成 SPFILE。

```
    CREATE    SPFILE=    'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
    FROM    PFILE    =    'C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
```

因为修改了 PFILE（INITZDB.ORA）中的参数，因此需要重建 SPFILE（SPFILEZDB.ORA）。

STEP 07 重启数据库。

```
SQL> STARTUP
```

11.4 查看控制文件的内容

可以把控制文件的内容 DUMP 到跟踪文件中，从跟踪文件中便可知道控制文件的内容，详细过程说明如下。

STEP 01 登录数据库。

```
SQL> CONNECT system@wm
```

STEP 02 获得 dump 文件的路径。

```
SQL>SHOW PARAMETER DIAGNOSTIC_DEST
```

SQL 语句的执行结果如下所示：

```
NAME                 TYPE        VALUE
------------------------------------ ------------------------------
diagnostic_dest      string
c:\app\administrator\diag\rdbms\zdb\zdb\trace
```

本例中，dump 文件的路径是 c:\app\administrator\diag\rdbms\zdb\zdb\trace。

STEP 03 DUMP 出控制文件的详细信息。

```
SQL> ALTER SESSION SET events 'immediate trace name controlf level 10';
Session altered.
```

STEP 04 到 c:\app\administrator\diag\rdbms\zdb\zdb\trace 下进行查找，找到 dump 文件，便可知控制文件的内容。

11.5 得到创建控制文件的脚本

通过执行 ALTER DATABASE BACKUP CONTROLFILE TO TRACE 命令，可以得到创建控制文件的脚本。

STEP 01 得到跟踪文件的路径。

```
SQL>SHOW PARAMETER user_dump_dest
```

SQL 语句的执行结果如下所示：

```
NAME                 TYPE        VALUE
------------------------------------ ------------------------------
user_dump_dest       string   c:\app\administrator\diag\rdbms\zdb\zdb\trace
```

本例中，跟踪文件的路径是 c:\app\administrator\diag\rdbms\zdb\zdb\trace。

STEP 02 把控制文件备份到数据库的跟踪文件中。

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
Database altered.
```

STEP 03 查看跟踪文件，得到创建控制文件的脚本。

11.6 移动控制文件

移动控制文件（Control Files）的过程和删除控制文件的过程很类似，都是通过修改初始化参数 control_files 来实现。具体过程说明如下（本例将移动控制文件 CONTROL03.CTL）。

STEP 01 以具有 SYSDBA 权限的用户身份登录数据库。

STEP 02 关闭数据库。

```
SQL> SHUTDOWN IMMEDIATE;
```

STEP 03 移动控制文件 CONTROL03.CTL。

本例中把控制文件 F:\NEWDB\ZDB\CONTROL03.CTL 移动到 D:\BC 之下。使用操作系统命令移动文件 CONTROL03.CTL 后，该文件的新路径是 D:\BC。

STEP 04 进入初始化参数文件所在的默认路径。

- ORACLE_HOME/dbs（Linux 和 UNIX 平台）
- ORACLE_HOME\database（Windows 平台）

其中，ORACLE_HOME 表示 Oracle 的主目录。

STEP 05 修改 PFILE（本例中是 INITZDB.ORA）。

编辑初始化参数文件 INITZDB.ORA，修改参数 control_files 把'D:\NEWDB\CONTROL03.CTL'替换成'D:\BC\CONTROL03.CTL'。即：

```
*.control_files='F:\NEWDB\ZDB\CONTROL01.CTL','F:\NEWDB\ZDB\CONTROL02.CTL', 'D:\BC\CONTROL03.CTL'
```

CONTROL03.CTL 的新路径是 D:\BC。

STEP 06 根据 PFILE 生成 SPFILE。

```
CREATE SPFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\SPFILEZDB.ORA'
FROM PFILE = 'C:\app\Administrator\product\11.1.0\db_3\database\INITZDB.ORA'
```

因为修改了 PFILE（INITZDB.ORA）中的参数，因此需要重建 SPFILE（SPFILEZDB.ORA）。

STEP 07 重启数据库。

```
SQL> STARTUP
```



11.7 删除控制文件

如果不再需要某个控制文件，可以将其删除。在 Oracle 中，删除控制文件的过程比较麻烦，不是用一个命令就能解决的问题。删除控制文件的过程说明如下。

STEP 01 以具有 SYSDBA 权限的用户身份登录数据库。

STEP 02 关闭数据库。

```
SQL> SHUTDOWN IMMEDIATE;
```

STEP 03 进入初始化参数文件所在的默认路径。

- ORACLE_HOME/dbs（Linux 和 UNIX 平台）
- ORACLE_HOME\database（Windows 平台）

其中，ORACLE_HOME 表示 Oracle 的主目录。

STEP 04 修改 PFILE（本例中是 INITZDB.ORA）。

编辑初始化参数文件 INITZDB.ORA，在参数 control_files 的后面删除 'D:\NEWDB\CONTROL04.CTL'，CONTROL04.CTL 是本例中要被删除的控制文件。即：

```
*.control_files='F:\NEWDB\ZDB\CONTROL01.CTL','F:\NEWDB\ZDB\CONTROL02.CTL', ' F:\NEWDB\ZDB\CONTROL03.CTL'
```

参数 control_files 已经不再含有 CONTROL04.CTL 的信息。

STEP 05 根据 PFILE 生成 SPFILE（这一步可选）。

```
CREATE SPFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\SPFILEZDB.ORA'
FROM PFILE = 'C:\app\Administrator\product\11.1.0\db_3\database\ INITZDB.ORA'
```

因为修改了 PFILE（INITZDB.ORA）中的参数，因此需要重建 SPFILE（SPFILEZDB.ORA）。

STEP 06 重启数据库。

```
SQL> STARTUP
```

STEP 07 使用操作系统命令彻底删除不再使用的控制文件 CONTROL04.CTL。



本章总结

本章介绍控制文件的概念和使用（包括镜像控制文件、移动控制文件、删除文件等），控制文件的重要性在本章中已经多次进行了强调。控制文件的损坏将直接影响到整个数据库，因此，在对控制文件进行维护时，需要我们百分百的小心！在一个系统中，应该镜像多个控制文件，并把它们分布在不同的物理磁盘，这是防止因为控制文件而导致系统崩溃最重要的举措！

第 12 章　服务器掉电亦无忧之——Oracle 重做日志（Redo Log）管理

一个漆黑的夜，狂风、暴雨……全城停电，一台高速运转的大型数据库服务器也戛然而止。公司所有的业务被迫停顿……几名工程师行色匆匆地赶到机房……来电了……他们给机器加电……重启数据库服务器……一切正常！

对于这样的事故，DBA 恐怕已习以为常。Oracle 有一套机制保证 DBA 高枕无忧。这其中最大的功劳非“重做日志”莫属！重做日志（Redo Log）保证在服务器掉电的情况下，也不会发生数据的丢失，这是重做日志的神奇之处！

12.1　重做日志文件的工作原理

接下来，我们将逐步揭开重做日志的神秘面纱。对于重做日志，我们必须清楚下面几个概念。

12.1.1　重做日志（Redo Log）

重做日志用于记录所有对数据库的改变。

12.1.2　重做日志条目（Redo Entry）

一个“重做日志条目”由一系列的改变因子（Change Vectors）组成。一个改变因子用于描述一次对某个数据块的改变。重做日志条目用于重构对数据库的改变，进行实例恢复时，Oracle 读取重做日志条目中的改变因子，然后把这些因子应用到相应的块上。

12.1.3　重做日志（Redo Log）的构成

如图 12-1 所示，一个实例有一组或者多组重做**日志组**（Redo Log Group）（也可叫“日

志组”)。每个重做日志组中有一个或者多个**重做日志文件**（Online Redo Log Files）（又叫“日志文件”或者“日志成员”)。每个实例至少需要两个**重做日志组**。同一组中每个重做日志文件的内容完全相同。

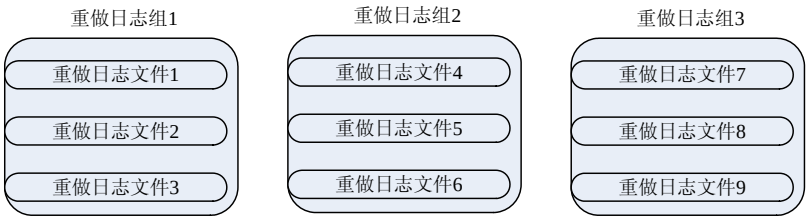


图 12-1　重做日志的组成

12.1.4　重做日志（Redo Log）的工作原理

后台进程 LGWR 负责把重做日志缓冲区中的重做日志条目写到重做日志文件中。当用户提交一个事务时，LGWR 把这个事务的重做条目写入重做日志文件，并赋予一个系统改变标识符（System Change Number，SCN），SCN 用于标识事务的重做日志条目。只有当事务的“重做日志条目”被成功写到磁盘上的重做日志文件中，事务的提交（COMMIT）才算完成。当然，在下面的情况下，LGWR 也会自动把重做日志缓冲区中的重做日志条目写到重做日志文件中：

- 每隔 3 秒钟
- 一个用户提交一个事务
- 日志缓冲区（Redo Log Buffer）被填满三分之一
- 日志缓冲区（Redo Log Buffer）有将近 1MB 的日志数据
- 发生校验点（Checkpoint）

如果当前的**重做日志组**被填满，Oracle 会切换到下一组**有效**的重做日志组，直到下一组重做日志组被填满，Oracle 又切换到另外一组**有效**的重做日志组。当所有的重做日志组都被填满后，Oracle 切换到第一组重做日志组继续写，如图 12-2 所示。

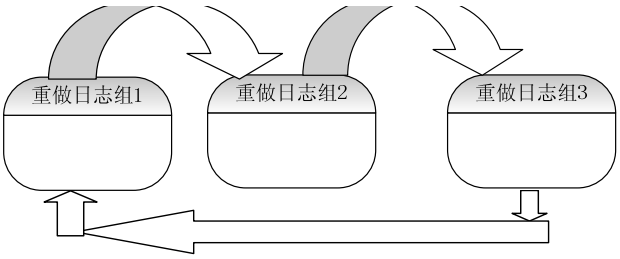


图 12-2　重做日志组的循环使用

进行日志组切换时，并不是所有的重做日志组都**有效**。如果数据库运行在非归档模式（NOARCHIVELOG）并且重做日志文件记录的内容已经写到数据文件中，则该重做日志组被认为**有效**；如果数据库运行在归档模式（ARCHIVELOG），重做日志文件记录的内容已

经写到数据文件中**并且已经归档**，则该重做日志组也被认为**有效**。有效的重做日志组可以被重新利用。

当一组重做日志组被填满时，Oracle 会自动进行日志切换（**Log Switches**），当然，我们也可以使用命令强迫 Oracle 进行日志切换。

12.1.5 重做日志文件（Redo Log Files）的四种状态

如果你想要全面了解 Oracle 的备份恢复机制，必须理解重做日志文件的四种状态。

重做日志文件有四种状态：CURRENT、ACTIVE、INACTIVE、UNUSED。

- CURRENT 表示正在被 LGWR 写的重做日志组的状态是 CURRENT。
- ACTIVE 表示重做日志组记录的改变还没有被写到数据文件中，实例恢复时（Instance Recovery）需要用到状态为 ACTIVE 的重做日志组。
- INACTIVE 表示实例恢复时（Instance Recovery）不需要状态为 INACTIVE 的重做日志组。
- UNUSED 表示该重做日志组是新建的，还未被使用。

12.1.6 同一日志组中成员的分布

重做日志文件（Online Redo Log Files）是非常重要的文件。同一个重做日志组中有部分文件损坏，数据库仍然能够继续工作。如果同一个重做日志组中所有的文件都损坏，数据库需要进行恢复。为了最大化地保护重做日志，同一重做日志组中应该有多个重做日志文件，并且每个重做日志文件应该分布在不同的物理磁盘。如图 12-3 所示，重做日志组 1 有 3 个重做日志文件（重做日志文件 1、重做日志文件 2、重做日志文件 3），每个重做日志文件分布在不同的物理磁盘：重做日志文件 1 分布在物理磁盘 A，重做日志文件 2 分布在物理磁盘 B，重做日志文件 3 分布在物理磁盘 C。这样可以实现对重做日志文件的最大化保护！

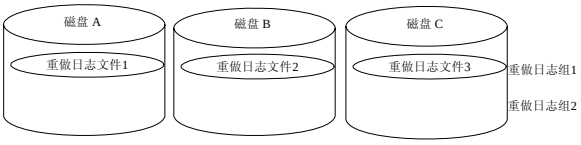


图 12-3 重做日志成员的分布

12.1.7 重做日志（Redo Log）与实例恢复

如果数据库突然掉电，对数据库的改变没有真正写到数据文件中，但是，Oracle 保证对数据库的改变已经被记录到重做日志文件（Online Redo Log Files）中。当我们重新启动数据库时，Oracle 将自动执行实例恢复，实例恢复会自动把记录在重做日志文件中的改变应用到数据文件中，就不会有数据的丢失。这就是我们所谓的“服务器掉电亦无忧”。

12.2 查询数据库中的重做日志组

查询视图 V$LOG，可以知道数据库中有哪些重做日志组，并且可以知道每个重做日志组的大小、成员、是否归档、状态等信息。

```
SELECT * FROM V$LOG;
```

SQL 语句的执行结果如下所示：

```
GROUP#  THREAD# SEQUENCE#   BYTES       MEMBERS         STATUS
1       1       79          52428800        1           NO      CURRENT
2       1       77          52428800        1           YES     INACTIVE
3       1       78          52428800        1           YES     INACTIVE
4       1       0           4194304         2           YES     UNUSED
```

本例中，共有 4 个重做日志组，第 4 组中有 2 个成员，其他组中只有一个日志成员。

- GROUP# 表示重做日志组的编号。
- THREAD# 表示线程号，线程与实例相关，不同的事例有不同的线程号。
- SEQUENCE# 表示日志序列号。
- BYTES 指定重做日志的大小。
- MEMBERS 指定有多少个成员。
- ARCHIVED 指定是否归档。
- STATUS 表示重做日志的状态。

12.3 查询数据库中的重做日志文件

从视图 V$LOGFILE 中可以知道数据库中有哪些重做日志文件，以及每个重做日志文件属于哪一组。执行下面的 SQL 语句，将会看到重做日志文件的信息。

```
SELECT * FROM V$LOGFILE;
```


SQL 语句的执行结果如下所示：

```
GROUP#  STATUS  TYPE        MEMBER              IS_RECOVERY_DEST_FILE
1               ONLINE      F:\NEWDB\ZDB\REDO01.LOG NO
2               ONLINE      F:\NEWDB\ZDB\REDO02.LOG NO
3               ONLINE      F:\NEWDB\ZDB\REDO03.LOG NO
4               ONLINE      F:\NEWDB\ZDBLOG1.RDO    NO
4               ONLINE      F:\NEWDB\ZDBLOG2.RDO    NO
```

- GROUP# 表示该文件属于哪一组。
- STATUS 表示日志成员的状态：
 - ✓ INVALID 表示文件不可访问；

- ✓ STALE 表示文件的内容不完整；
- ✓ DELETED 表示文件不再被使用；
- ✓ NULL 表示文件在使用中。

➢ TYPE 表示文件的类型（ONLINE、STANDBY）。

➢ MEMBER 表示重做日志组成员（文件）的名字及路径。

12.4 创建重做日志组（Creating Redo Log Groups）

如果数据库中日志切换频繁，建议增加重做日志组。增加重做日志组的方法说明如下。

STEP 01 创建重做日志组。

```
ALTER DATABASE
ADD LOGFILE ('F:\newdb\zdblog1.rdo', 'F:\newdb\zdblog2.rdo') SIZE 4M;
```

往数据库中添加一个重做日志组，该日志组中共有两个重做日志文件：'F:\newdb\zdblog1.rdo'、F:\newdb\zdblog2.rdo'，文件的大小是 4MB。值得注意的是，同一组中的重做日志文件具有相同的尺寸。

STEP 02 创建重做日志，并指定重做日志组的编号是 10。

```
ALTER DATABASE
ADD LOGFILE GROUP 10 ('F:\newdb\zdblog1.rdo', 'F:\newdb\zdblog2.rdo') SIZE 20M;
```

每个重做日志组都有一个编号，只是有的编号是我们手工指定的，有的编号是系统自动产生的（如果我们不指定的情况下）。

12.5 添加重做日志文件（Creating Redo Log Members）

我们可以往已经存在的**重做日志组**中添加**重做日志文件**以实现对重做日志的最大保护。添加重做日志文件的命令如下所示：

```
ALTER DATABASE ADD LOGFILE MEMBER 'F:\newdb\zdblog5.rdo' TO GROUP 2;
```

往重做日志组 **2** 中添加成员 F:\newdb\zdblog5.rdo。记住，不能对单个成员指定大小。

12.6 删除重做日志组（Dropping Redo Log Groups）

删除**重做日志组**操作只是更新了控制文件，被删除的组拥有的重做日志文件并没有真正从操作系统被删除，DBA 需要使用操作系统命令手工删除磁盘上的重做日志文件，并且只能删除状态是 Inactive 的重做日志组。如果想删除当前（Current）重做日志组，需要强迫日志（使用命令 ALTER SYSTEM SWITCH LOGFILE）。每个数据库至少需要两组重做日志组。删除重做日志组的过程说明如下。

STEP 01 查询重做日志组的状态。

```
SELECT GROUP#, ARCHIVED, STATUS FROM V$LOG;
```

STEP 02 删除重做日志组。

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

本例中，删除重做日志组 3。

STEP 03 使用操作系统命令删除磁盘上的重做日志文件。

12.7 删除重做日志成员（Dropping Redo Log Members）

可以删除重做**日志成员**（也就是重做日志文件），但是一个日志组中至少需要保留一个成员。只能删除状态不是 ACTIVE 或 CURRENT 的重做日志成员。

STEP 01 查询重做日志文件。

```
SELECT * FROM V$LOGFILE;
```

STEP 02 删除重做日志文件。

```
ALTER DATABASE DROP LOGFILE MEMBER 'F:\NEWDB\ZDBLOG2.RDO';
```

本例中删除重做日志文件 F:\NEWDB\ZDBLOG2.RDO。

12.8 改变重做日志组的状态

如果重做日志组的状态是 CURRENT，则执行下面的命令，强迫日志切换。

```
ALTER SYSTEM SWITCH LOGFILE;
```

命令执行完成后，该日志组的状态将变成其他状态。

如果重做日志组的状态是 ACTIVE，执行下面的命令，日志组的状态将变成 INACTIVE。

```
ALTER SYSTEM CHECKPOINT;
```



知识点索引——日志切换（Log Switch）

日志切换（Log Switch）就是重做日志书写进程 LGWR 停止写当前重做日志组，切换到下一日志组，开始写下一日志组的过程。在一般情况下，日志的切换由 Oracle 自动完成。有时候，我们也可以使用 ALTER SYSTEM SWITCH LOGFILE 命令强迫 Oracle 进行日志切换（即使当前日志组还没被写满）。



12.9 查看重做日志组是否已经归档

通过视图 v$log，可以知道重做日志组是否归档，具体过程如下：

```
SELECT group#,status,archived
FROM v$log;
```

SQL 语句的执行结果如下所示：

```
GROUP#  STATUS          ARCHIVED
---------- ---------------- --------
       1       INACTIVE           YES
       2       CURRENT            NO
       3       INACTIVE           YES
       4       INACTIVE           NO
```

ARCHIVED=YES 表示日志已经归档，ARCHIVED=NO 表示未归档。

从上面的结果可以看出，日志组 1（GROUP 1）已经归档，日志组 4（GROUP 4）还未归档。



本章总结

本章主要介绍重做日志文件的原理。如果一个重做日志组中所有的重做日志文件坏掉，Oracle 可能面临崩溃的风险！因此，维护重做日志时，一定要确保每一个**重做日志组**中至少有**两个重做日志文件**，并且把这些文件分布到**不同**的物理磁盘。重做日志文件是写最频繁的文件，如果你是 DBA，应该随时关注重做日志文件对数据库性能的影响，最好把重做日志文件放到高速磁盘上。

第 13 章　总揽全局——Oracle 体系结构综述

谢谢大家能够陪伴我们走到第 13 章，遨游在数据库的海洋中，我们好似海滩边捡贝壳的小孩子，今天看到一个小贝壳，漂洗一下，收入囊中；明天发现镶嵌在泥中的小贝壳，总是刨根问底；有时在海边漫步，不轻易间踢到被别人丢弃的小贝壳，我们也禁不住驻足观看。偶尔，看到小贝壳在蔚蓝的大海中随波逐流，我们也会不假思索地去追逐和探索！我们需要冒险，我们需要创新，我们需要改变！

在以前的学习中，我们收集了太多零零散散的小贝壳，散而无用！只有把这些小贝壳串联起来，深度加工，才能体现它们的价值。学习任何技能也都如此，只有融会贯通，才能技高一筹！不过，这只是我的愚见！

前面的章节，我们介绍了数据库、实例、SGA 的概念，还介绍了 Oracle 的物理结构、逻辑结构和后台进程。接下来，我们将从宏观到微观、从总体到局部梳理一下 Oracle 的体系结构。

如图 13-1 所示，一台 Oracle 数据库服务器（Oracle Database Server）由一个 Oracle 数据库（Oracle Database）和一个或者多个数据库实例（Oracle Database Instances）组成。

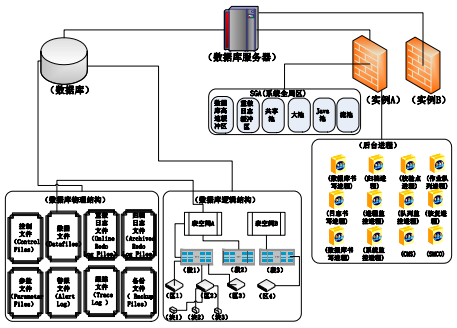


图 13-1　Oracle 体系结构综述

一个数据库有物理结构（Physical Database Structures）和逻辑结构（Logical Database Structures）。一个实例由 SGA 和一系列后台进程组成。

13.1 物理数据库结构（Physical Database Structures）

Oracle 的物理结构（Physical Database Structures）包括：

- 数据文件（Datafiles）
- 控制文件（Control Files）
- 重做日志文件（Online Redo Log Files）
- 归档日志文件（Archived Redo Log Files）
- 参数文件（Parameter Files）
- 警报文件（Alert Log Files）
- 跟踪文件（Trace Log Files）
- 备份文件（Backup Files）

值得一提的是，连接物理结构（Physical Structures）和逻辑结构（Logical Structures）之间的纽带是**表空间**和**数据文件**，一个或者多个数据文件构成表空间。表空间和数据文件的关系，**请参考本书第 7 章的内容（也可参考创建表空间的 SQL 语句）**。

13.2 逻辑数据库结构（Logical Database Structures）

逻辑数据库结构（Logical Database Structures）包括：

- 数据块（Oracle Database Data Blocks，也叫“块”）
- 区（Extents）
- 段（Segments）
- 表空间（Tablespaces）

一个区（Extent）由多个块（Oracle Database Data Block）组成， 多个区组成段，段存放在表空间中（创建表或者索引时，指定存放的表空间）。一个数据库由多个表空间构成。

13.3 实例（Database Instance）

后台进程（Background Processes）和 SGA 组成一个数据库实例（Database Instance）。

13.3.1 SGA

SGA 叫系统全局区（System Global Area），是一片共享的内存区域。SGA 包括以下几片内存区域：

- 数据库高速缓冲区（Database Buffer Cache）
- 重做日志缓冲区（Redo Log Buffer）
- 共享池（Shared Pool）
- 大池（Large Pool）
- Java 池（Java Pool）
- 流池（Streams Pool）

13.3.2 后台进程（Background Processes）

Oracle 的后台进程包括：

- 归档进程（Archiver Processes，ARCn）
- 校验点进程（Checkpoint Process，CKPT）
- 数据库书写进程（Database Writer Process，DBWR）
- 作业队列进程（Job Queue Process）
- 日志书写进程（Log Writer Process，LGWR）
- 进程监控进程（Process Monitor Process，PMON）
- 队列监控进程（Queue Monitor Process，QMNn）
- 恢复进程（Recover Process，RECO）
- 系统监控进程（System Monitor Process，SMON）
- 其他后台进程（Other Background Process）

13.3.3 最容易混淆的两个概念——数据库和实例

“数据库”和“实例”，这是最容易混淆的两个概念。在讲述数据库和实例的关系之前，我们必须首先弄清楚什么是数据库、什么是实例。

数据库（Database）

一个数据库（Database，简称 DB）是一系列数据的集合。它按照一定的方式存储并检索、利用数据。

实例（Instance）

一系列的**后台进程**（Background Processes）和**内存**（Memory Structures）组成一个 Oracle 实例（Oracle Instance），如图 13-2 所示，这里特别强调的是两者的结合。启动实例时，如

果你仔细观察，将会看出 Oracle 会分配内存，并启动一些后台进程。实例是动态的，只有数据库启动的时候，实例才存在；一旦数据库关闭，实例将随之消失。**有关 Oracle 内存的介绍请参考本书第 9 章的内容；后台进程的介绍请参考本书第 10 章的内容。**



图 13-2　实例的组成

数据库和实例的关系

对于第一次接触“实例”这个概念的朋友来说，容易把“实例”和“数据库”这两个概念混淆，实例（Instance）是“内存”和“后台进程”的集合。“数据库”是数据的物理存储。我们要特别注意的是，一个实例可以用于驱动一个数据库，多个实例也可同时驱动一个数据库（在 Oracle 应用集群环境中），实例和数据库的关系是多对一的关系。实例和数据库的关系如图 13-3 所示。

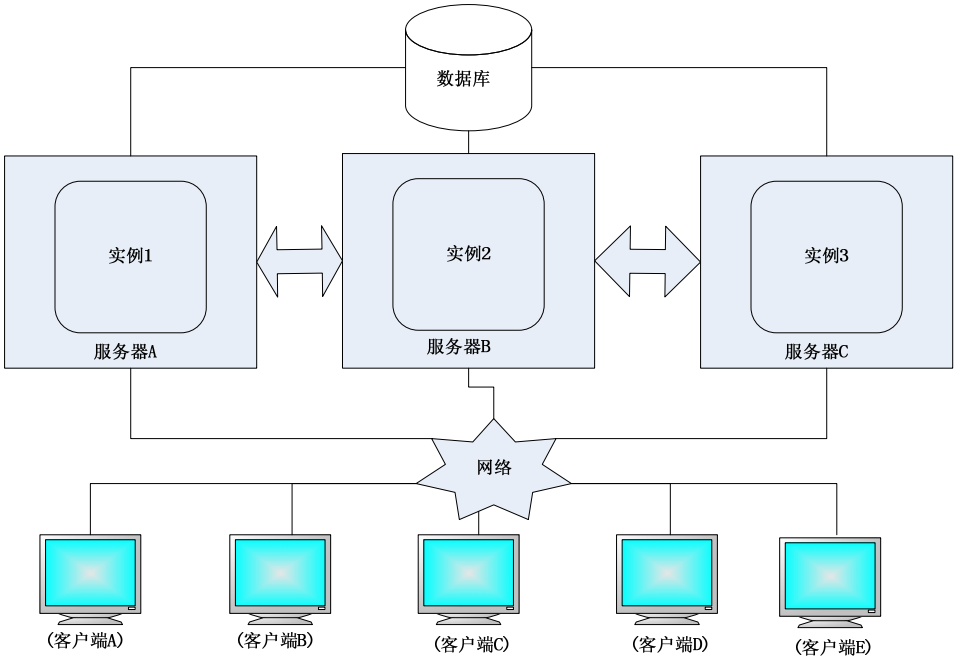


图 13-3　数据库和实例的关系

为了让大家更好地理解“数据库”和“实例”的关系，我再举一个例子。如果把“数据库”比作一架飞机，“实例”就是飞机的发动机。那么，一台发动机可以驱动一架飞机，多台发动机也可以同时驱动一架飞机。

从图 13-3 可以看出，多个实例共享一个数据库。数据通常存放在共享存储上。每个实例通常运行在一台独立的服务器上（硬件独立），实例之间需要相互通信和交换数据。客户端通过与实例交互实现对数据的访问。



13.4 阅读参考

为了更好地融会 Oracle 的各个知识点，贯通 Oracle 的体系结构，你需要参考相关章节，以下是阅读参考内容。

- 块（Data Blocks），**请参考本书 2.2.3 节的内容**。
- 警报文件（Alert Log Files），**请参考本书 6.2 节的内容**。
- 跟踪文件（Trace Log Files），**请参考本书 6.2 节的内容**。
- 数据块（Data Block）、区（Extent）、段（Segment）和表空间（Tablespace），**请参考本书 7.1.10 节的内容**
- 表空间（Tablespace），**请参考本书 7.1.1 节的内容**。
- 数据文件（Datafiles），**请参考本书 7.1.1 节的内容**。
- 数据库（Database）的概念，**请参考本书第 1 章的内容**。
- 内存（Memory）、SGA、PGA 的概念，**请参考本书第 9 章的内容**。
- 后台进程（Background Processes），**请参考本书第 10 章的内容**。
- 参数文件（Parameter Files），**请参考本书 6.1.2 节的内容**。
- Oracle 控制文件（Control Files），**请参考本书第 11 章的内容**。
- 重做日志（Online Redo Log Files），**请参考本书第 12 章的内容**。
- 归档日志文件（Archived Redo Log Files），**请参考本书第 22 章的内容**。



本章总结

本章疏通了 Oracle 的体系结构。应当仔细阅读，因为它让 Oracle 的框架结构一目了然。无论你是在学习以前的章节，还是后续的章节，建议大家多看看图 13-1，只有从面到点，然后从点到面，不断往复，我们才能实现对知识的融会贯通！

第 14 章　成为数据库的王者之——权限管理

若想在数据库中无拘无束地翱翔，必须取得数据的控制权，本章将和大家一起探讨如何成为“数据库的王者”！

14.1　用户管理



本节将要介绍 Oracle 中用户的概念。

14.1.1　创建用户账号



在介绍数据库用户之前，我们先体会一下如何缔造 2 个用户 nokey 和 itme。

■ 示例 1：创建用户（简化版）

```
CREATE USER nokey
IDENTIFIED BY zero;
```

本例使用最简单的 SQL 语句创建了用户 nokey。

■ 示例 2：创建用户（烦琐版）

```
CREATE USER itme
IDENTIFIED BY zero
DEFAULT TABLESPACE MYSPACE
QUOTA 10M ON MYSPACE
QUOTA 5M ON system
TEMPORARY TABLESPACE temp
PROFILE DEFAULT
PASSWORD EXPIRE
ACCOUNT UNLOCK;
```

本例使用较复杂的 SQL 语句创建用户 itme。“烦琐版”为用户指定了更多的选项。



知识点索引——用户账户

为了访问数据库，必须创建一个账号，并授予该账号一些权限。在使用数据库之前，用户必须先连接到数据库。每个账号都有一些属性，这些属性是：

- ✓ 账号名（用户名）。如例子中的 nokey 和 itme
- ✓ 密码（口令）。例子中，用户 nokey 和 itme 的密码都是 zero，特别注意的是，在 Oracle 11g 中，密码大小写敏感！IDENTIFIED BY 用于指定用户的口令。
- ✓ 默认表空间（Default Tablespace）。示例 2 中，用户 itme 的默认表空间是 MYSPACE。默认表空间是一个表空间。每个数据库用户被赋予一个默认表空间。当该用户创建数据库对象（如表）时，如果用户没有在 SQL 语句中指定对象放在哪个表空间，则对象将被 Oracle 存放在用户的默认表空间。切记，默认表空间是数据库用户的属性之一。关键字 DEFAULT TABLESPACE 用于指定用户默认表空间。
- ✓ 表空间限额（Tablespace Quota）。如示例 2 所示，用户 itme 有表空间 MYSPACE 上 10MB 空间的使用权限以及表空间 system 上 5MB 空间的使用权限。空间限额用于限制用户对表空间的使用。DBA 可以限制每个用户在每个表空间上使用空间的大小，如限制用户 A 只能使用表空间 TBS1 中 50MB 的空间，当用户 A 用完这 50MB 的空间以后，将不能在这个表空间上创建对象（即使这个表空间上还有大量空余空间）。表空间限额的目的是限制某些用户对表空间的过度消耗。QUOTA …ON 用于指定表空间限额。如果不指定，则该用户没有表空间的使用权限（除非已经被授予系统权限 UNLIMITED TABLESPACE）。
- ✓ 临时表空间（Temporary Tablespace）。每个用户有一个临时表空间。当用户执行 SQL 语句，且这些 SQL 语句涉及排序（如 ORDER BY）操作时，Oracle 将在指定的临时表空间（如示例 2 中，用户 itme 的临时表空间是 temp）中创建临时段（Temporary Segments），临时段用于排序。理想状态下，每个用户应该有一个单独的临时表空间，这样可以减少临时段对磁盘的竞争，以提高数据库的性能。多个用户可以共享一个临时表空间。
- ✓ 用户概要文件（User Profiles），示例 2 中，概要文件 DEFAULT 表示使用默认的概要文件。概要文件用于限制用户对资源的使用。PROFILE 是关键字，用于指定概要文件。如果没有指定概要文件，Oracle 会给该用户指定一个默认的概要文件。有关概要文件，将在后续章节中进行介绍。
- ✓ 密码期限。PASSWORD EXPIRE 指定该用户是否过期，如果指定该子句，在登录前，该用户或者 DBA 必须修改该用户的密码。
- ✓ 账号状态。示例 2 中，账号 itme 创建完成以后，处于非锁定状态。ACCOUNT 用于指定账号的状态。



知识点索引——数据库用户和模式的区别（Database Users and Schemas）

一个模式（Schema）是一系列的数据库对象（表、索引、视图等），一个模式被一个数据库用户所拥有，模式的名字和数据库用户的名字相同，这是模式和数据库用户的关系。通俗一点理解，用户名就是模式名，模式名就是用户名。



14.1.2 预定义账户（Predefined User Accounts）

除了我们创建的账户，在创建数据库时，Oracle 自动创建了一些账户，我们把这些账户叫做预定义账户（Predefined User Accounts），有三种类型的预定义账户：

- 管理员账户（Administrative Accounts）。这些账户包括 SYS、SYSTEM、SYSMAN、DBSNMP。SYSMAN 用于执行企业管理（OEM）管理任务。DBSNMP 用于监控数据库。这些用户与生俱来，不能随意删除这些用户！
- 样例账户（Sample Schema Account），为了学习的方便，在 Oracle 中，存在一些样例数据，Oracle 创建相应的账户（HR、SH、OE）管理样例数据。我们可以使用这些账户登录数据库。
- 内部账户（Internal Account），Oracle 的一些新特性或者一些组件将会使用这些账户。我们不能删除这些账户，也不能使用这些账户登录数据库。WKSYS 是内部账户之一。

重要账户 SYS

SYS 在数据库创建时被自动创建，并被授予 DBA 的角色，SYS 的默认密码是 CHANGE_ON_INSTALL。数据库字典涉及的所有基表和视图都放在该用户下。不要在该模式下创建其他的数据库对象。

重要账户 SYSTEM

SYSTEM 也是在数据库创建时被自动创建的，并被授予 DBA 的角色。SYSTEM 的默认密码是 MANAGER。数据库管理员通常使用该账号管理数据库。



知识点索引——DBA 角色

DBA 角色是预定义的角色，在数据库创建的时候被创建。DBA 角色含有大量的数据库系统权限，因此，不要把该角色授予普通的用户。虽然该角色拥有大量的数据库系统权限，但它还是没有 SYSDBA、SYSOPER 两项系统权限。



14.1.3 修改用户账号属性

使用 ALTER USER 命令可以修改用户的属性，如密码、用户默认表空间、临时表空间等。

■ 修改用户密码

```
ALTER USER itme
IDENTIFIED BY zero2
```

将用户 itme 的密码改成 zero2。

■ 使用户的密码过期

```
ALTER USER itme PASSWORD EXPIRE;
```

■ 修改用户的临时表空间

```
ALTER USER itme
TEMPORARY TABLESPACE temp;
```

把用户 itme 的临时表空间改成 temp。

■ 修改用户的默认表空间

```
ALTER USER itme
DEFAULT TABLESPACE users;
```

把用户 itme 的默认表空间改成 users。

■ 修改用户的表空间限额

```
ALTER USER itme
QUOTA 50M ON users;
```

使用户 itme 只有表空间 users 上 50MB 的使用权限。



14.1.4 锁定爱捣蛋的用户

如果某个用户在数据库中胡作非为，又不好立马删除该用户，可以先锁住该用户，再寻找对策。锁住用户的 SQL 语句如下所示：

```
ALTER USER itme ACCOUNT LOCK;
```

可以使用下面的命令进行解锁。

```
ALTER USER itme ACCOUNT UNLOCK
```




14.1.5 查看数据库账户属性

查询视图 DBA_USERS，可以知道账户的信息，如账户的临时表空间、默认表空间等信息。

■ 查看单个用户的信息

```
SELECT USERNAME, PROFILE, ACCOUNT_STATUS FROM DBA_USERS
WHERE USERNAME='ITME';
```

SQL 语句的执行结果如下所示：

```
USERNAME        PROFILE         ACCOUNT_STATUS
ITME            TEST_PROFILE    LOCKED
```

本例查看用户 ITME 的信息。

■ 查看所有的用户信息

```
SELECT USERNAME, PROFILE, ACCOUNT_STATUS FROM DBA_USERS
```

■ 得到所有用户账号的信息

```
SELECT * FROM DBA_USERS
```

■ 查看用户的表空间限额

```
SELECT * FROM DBA_TS_QUOTAS;
```

SQL 语句的执行结果如下所示：

```
TABLESPACE_NAME USERNAME        BYTES   MAX_BYTES   BLOCKS  MAX_BLOCKS
DROPPED
SYSAUX          OLAPSYS     16318464    -1          1992        -1
NO
SYSAUX          WK_TEST     14155776    -1          1728        -1
NO
MYSPACE         ITME        0           10485760        0       1280
    NO
SYSTEM          ITME        0           52428800        0       6400
    NO
```

用户 OLAPSYS 可以无限制使用表空间 SYSAUX（MAX_BYTES=-1）；用户 ITME 可以使用表空间 MYSPACE 上 10485760 字节的空间。

■ 查看每个会话内存的使用情况

```
SELECT USERNAME, VALUE || 'bytes' "Current UGA memory"
FROM V$SESSION sess, V$SESSTAT stat, V$STATNAME name
WHERE sess.SID = stat.SID
AND stat.STATISTIC# = name.STATISTIC#
AND name.NAME = 'session uga memory';
```

SQL 语句的执行结果如下所示：

```
USERNAME        Current UGA memory
SYSTEM          629540bytes
SYSTEM          825836bytes
SYSTEM          694908bytes
ITME            236660bytes
```

从上面可以看出，有关用户 ITME 的其中一个会话消耗 236660 字节的内存。

14.1.6 让一个用户从数据库中永远消失

删除一个用户将从数据字典中删除该用户及该用户下的模式对象。删除用户的完整过程说明如下。

STEP 01 找出和用户相关的会话。

```
SELECT SID, SERIAL#, USERNAME FROM V$SESSION;
```

SQL 语句的执行结果如下所示：

```
SID SERIAL# USERNAME
------- --------------- ----------------------
133 45257 nokey
...
```

不能删除正在连接数据库的用户，所以必须找出这些会话，终止这些会话。本例中要删除的用户是 nokey。

STEP 02 终止要被删除的用户相关的会话。

```
ALTER SYSTEM KILL SESSION '133, 45257';
```

STEP 03 查看该用户拥有的模式对象。

```
SELECT OWNER, OBJECT_NAME FROM DBA_OBJECTS WHERE OWNER LIKE ' nokey';
```

删除该用户后，该用户下的模式对象将消失。

STEP 04 删除用户。

```
SQL> DROP USER nokey;
User dropped
```

如果用户模式下存在对象，则使用下面的语句：

```
SQL> DROP USER nokey CASCADE;
User dropped
```

其中，CASCADE 表示在删除用户之前，将首先删除用户模式下的对象。

14.1.7 查看当前登录数据库的用户

查询视图 V$SESSION，可知有哪些已经用户登录数据库。执行下面的 SQL 语句：

```
SQL>SELECT USERNAME FROM V$SESSION WHERE TYPE='USER';
```

SQL 语句的执行结果如下所示：

```
USERNAME
SYSTEM
ITME
```

本例中，已经登录数据库的用户有 SYSTEM 和 ITME。

14.2 限制用户对资源的消耗——概要文件（User Profiles）

用户概要文件（User Profiles，也叫资源限制文件）用于限制用户对数据库资源的消耗及账号的口令策略。数据库的资源包括：

- CUP 时间（CUP Time）
- 会话的空闲时间（Idle Time For A Session）
- 会话的连接时间（Connect Time）
- 逻辑读（Logical Reads）
- 每个用户的并发会话数（Concurrent Sessions For Each User）
- 私有 SGA 区及用户私有 SQL 区（Private SQL Areas）

14.2.1 创建概要文件

在把概要文件赋予用户之前，需要创建概要文件，本文将通过两个例子讲解如何创建概要文件。

■ 示例 1：创建概要文件 test_profile

```
CREATE PROFILE test_profile LIMIT
SESSIONS_PER_USER UNLIMITED
CPU_PER_SESSION UNLIMITED
CPU_PER_CALL 3000
CONNECT_TIME 45
LOGICAL_READS_PER_SESSION DEFAULT
LOGICAL_READS_PER_CALL 1000
PRIVATE_SGA 15K
COMPOSITE_LIMIT 5000000;
```

概要文件的参数说明如下：

- test_profile 表示概要文件的名字。
- SESSIONS_PER_USER 表示每个用户允许建立的最大会话数，值 UNLIMITED 表示无限制。
- CPU_PER_SESSION 指定每个会话的 CPU 时间限制，单位是百分之一秒，UNLIMITED 表示无限制。
- CPU_PER_CALL 表示每次调用最多可以消耗的 CUP 时间，单位是百分之一秒。
- CONNECT_TIME 表示每个会话能够连接到数据库的最长时间，单位是分钟。如果一个会话超过连接时间，Oracle 会回滚任何未提交的数据，然后中止该会话。如果设置成 UNLIMITED，该用户将可以无限期连接数据库。

- LOGICAL_READS_PER_CALL 表示每个调用所能读取的数据块的最大数量。
- LOGICAL_READS_PER_SESSION 表示每个会话所能读取的数据块的最大数量。
- PRIVATE_SGA 表示在 SGA 中能为每个会话分配的私有 SQL 区（Private SQL Area）的最大值，单位是 KB 或 MB。
- COMPOSITE_LIMIT 表示一个会话所消耗资源的总额度。

■ 示例 2：创建概要文件 profile_password（该文件用于设定用户的口令策略）

```
CREATE PROFILE profile_password LIMIT
FAILED_LOGIN_ATTEMPTS 5
PASSWORD_LIFE_TIME 60
PASSWORD_REUSE_TIME 60
PASSWORD_REUSE_MAX 5
PASSWORD_LOCK_TIME 1/24
PASSWORD_GRACE_TIME 10;
```

概要文件的参数说明如下：

- FAILED_LOGIN_ATTEMPTS 指定允许登录失败的次数，超过指定的次数后，这个账号将被锁定。
- PASSWORD_LIFE_TIME 指定口令的生存时间，单位是天。如果在口令的生存时间内，用户没有修改过口令，则这个口令在生存时间到达之后将会失效。
- PASSWORD_REUSE_TIME 表示用户重复使用一个口令之前必须经历的时间，单位是天。
- PASSWORD_REUSE_MAX 表示用户在重用一个口令之前必须修改的次数。
- PASSWORD_LOCK_TIME 指用户由于口令输入错误口令被锁住，保持锁住状态的时间。
- PASSWORD_GRACE_TIME 指定用户口令过期的时间，如果达到这个期限，用户还没有修改口令，将会收到警告信息。



14.2.2 修改用户的概要文件（把概要文件赋予用户）

使用 ALTER USER 命令把创建好的概要文件赋予用户，以限制该用户对数据库资源的使用。

```
ALTER USER itme
PROFILE test_profile;
```

本例把概要文件 test_profile 赋予用户 itme。



14.2.3 删除概要文件

使用命令 DROP PROFILE 删除概要文件，删除概要文件需要用户具有 DROP PROFILE 权限。

```
DROP PROFILE test_profile CASCADE;
```

本例删除概要文件 test_profile。

如果概要文件已经被赋予用户，则需要加上选项 CASCADE 才能删除概要文件。



14.3 数据库管理员的验证方式（Authentication Method for Database Administrators）

数据库管理员执行数据库的重大操作（如启动和关闭数据库），这些数据库管理员并不是拥有 DBA 角色的用户，而是拥有 SYSDBA、SYSOPER 权限的用户。这些用户的权力巨大。为了对数据库进行维护，这些用户需要登录数据库（即使数据库没有启动）。登录数据库时，Oracle 需要对这些权力用户进行特殊的验证。Oracle 提供了 4 种方法进行验证：数据字典验证（Database Data Dictionary Authentication）、操作系统验证（Operating System Authentication）、口令文件验证（Password File）和基于网络的验证，如 Oracle 互联网目录（Oracle Internet Directory）。



14.3.1 管理员的验证之——数据字典验证（Data Dictionary Authentication）

数据字典验证（Data Dictionary Authentication）方式，又叫数据库验证方式（Database Authentication），通过存储在数据库中的信息对用户进行验证。要使用数据字典验证方式，必须创建带有密码的数据库用户。建立数据库连接时，必须提供用户名和密码。



14.3.2 管理员的验证之——使用操作系统验证（Using Operating System Authentication）

使用操作系统验证，即在不需要用户名和密码的情况下也能登录数据库。使用操作系统验证时，两个操作系统组（OSDBA 和 OSOPER）控制数据库管理员的连接。组 OSDBA 和 OSOPER 的具体名字根据平台的不同而有所不同（如表 14-1 所示）。

表 14-1 操作系统组

操作系统组	UNIX 平台	Windows 平台
OSDBA	dba	ORA_DBA
OSOPER	oper	ORA_OPER

如果一个用户属于操作系统组 OSDBA，则该用户可以使用 AS SYSDBA 登录数据库。

如果一个用户属于操作系统组OSOPER,则该用户可以使用 AS SYSOPER登录数据库。

使用操作系统验证的实现步骤说明如下。

STEP 01 创建一个操作系统用户。

STEP 02 把该用户添加到 OSDBA 或者 OSOPER 组。

STEP 03 打开 SQL*Plus。

在操作系统命令行中输入：

```
sqlplus /NOLOG
```

STEP 04 登录数据库。

```
CONNECT / AS SYSDBA
```

或

```
CONNECT / AS SYSOPER
```

如果登录远程数据库，可以使用下面的方法：

```
CONNECT /@net_service_name AS SYSDBA
```

或

```
CONNECT /@net_service_name AS SYSOPER
```

其中，net_service_name 是网络服务名。



注：这种方法只适用于客户端和数据库服务器都属于一个 Windows 域（Windows Domain）的环境。



14.3.3 管理员的验证之——使用口令文件验证（Using Password File Authentication）

Oracle 使用口令文件（Password Files，密码文件）来跟踪具有 SYSDBA 及 SYSOPER 权限的用户。口令文件位于数据库之外。如果使用口令文件验证，当数据库管理员试图建立数据库连接时，必须通过口令文件验证。口令文件是一个操作系统文件，用于存放被授予 SYSDBA 及 SYSOPER 权限的用户的用户名和密码。它是一个加密的操作系统文件，用户不能手工修改密码文件，也不能直接从口令文件中获得超级用户的用户名和密码。

Oracle 使用口令文件创建工具 ORAPWD 创建口令文件。ORAPWD 是 Oracle 提供的一个小工具，在操作系统中运行。创建口令文件的语法如下所示：

```
ORAPWD FILE=filename [ENTRIES=numusers]
[FORCE={Y|N}] [IGNORECASE={Y|N}] [NOSYSDBA={Y|N}]
```

- FILE 参数指定口令文件存放的全路径及文件名。口令文件的命名也应该遵循 Oracle 公司指定的规范。口令文件的命名是 PWDsid.ORA，口令文件的默认路径是 ORACLE_BASE\ORACLE_HOME\DATABASE（Windows 平台）。

- ENTRIES 为整数值，指定最多可以有多少个用户成为超级用户。
- FORCE 取值为“Y”或者“N”，指定是否强行覆盖已经存在的口令文件。
- IGNORECASE=Y　表示口令大小写不敏感。

口令文件验证的实现

本文将通过一个例子，讲述如何实现口令文件验证。本例也可叫“如何使普通用户成为超级用户”。具体过程说明如下。

STEP 01 创建口令文件。

```
    orapwd FILE=C:\app\Administrator\product\11.1.0\db_3\database\PWDzdb.ORA
ENTRIES=5 FORCE=y
```

本例中创建口令文件 PWDzdb.ORA。

创建口令文件的过程中，如果提示错误信息“OPW-00001: 无法打开口令文件”，可能是由于引号引起的错误，去掉引号即可。

STEP 02 进入初始化参数文件所在的目录。

初始化参数文件（PFILE）和服务器参数文件（SPFILE）位于以下的路径。

- ORACLE_HOME/dbs（Linux 和 UNIX 平台）
- ORACLE_HOME\database（Windows 平台）

其中，ORACLE_HOME 表示 Oracle 的主目录。

STEP 03 如果数据库使用的是 SPFILE，则根据 SPFILE 生成初始化参数文件（PFILE），否则直接执行下一步。

```
    SQL> CREATE PFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
      2  FROM  SPFILE = 'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
      3 /
```

文件已创建

其中，SPFILEZDB.ORA 是 SPFILE，INITZDB.ORA 是 PFILE。

STEP 04 编辑初始化参数文件（PFILE）。

编辑初始化参数文件 INITZDB.ORA，将 REMOTE_LOGIN_PASSWORDFILE 的值改成 EXCLUSIVE。REMOTE_LOGIN_PASSWORDFILE 的值说明如下：

- NONE 表示未使用口令文件验证；
- EXCLUSIVE 表示一个口令文件只能被一个实例使用；
- SHARED 表示一个口令文件可以被多个实例使用。

STEP 05 关闭数据库。

```
    SQL> SHUTDOWN IMMEDIATE;
```

数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。

STEP 06 重建 SPFILE。

```
    SQL> CREATE SPFILE= 'C:\app\Administrator\product\11.1.0\db_3\database\
SPFILEZDB.ORA'
```

```
    2   FROM  PFILE  =  'C:\app\Administrator\product\11.1.0\db_3\database\
INITZDB.ORA'
    3  /
    文件已创建
```

STEP 07 重启数据库。

```
    SQL>STARTUP
```

STEP 08 以 SYS 登录数据库。

STEP 09 创建新的用户。

本例中创建用户 newsuper。

```
    CREATE USER newsuper IDENTIFIED BY zero;
```

STEP 10 把权限 SYSDBA 及 SYSOPER 授予新建的用户。

```
    GRANT SYSDBA TO newsuper;
```

STEP 11 测试新用户可不可以登录。

```
    CONNECT newsuper  AS SYSDBA
```

用户 newsuper 能够以 SYSDBA 的身份登录数据库，说明普通用户 newsuper 已经成为超级用户，登录数据库后，newsuper 就可以执行数据库的重大操作了（如启动和关闭数据库）。

查询数据库中的超级用户

查询视图 V$PWFILE_USERS，可以知道数据库中有哪些超级用户（具有 SYSDBA 或者 SYSOPER 权限的用户）。

```
    SQL> SELECT * FROM V$PWFILE_USERS;
```

SQL 语句的执行结果如下所示：

```
    USERNAME                       SYSDB   SYSOP SYSAS
    ------------------------------ ----- ----- -----
    SYS                            TRUE      TRUE  FALSE
    NEWSUPER                       TRUE     FALSE FALSE
```

如果 SYSDB 或者 SYSOP 的值等于 TRUE，说明这个用户是超级用户。本例中共有两个超级用户：sys 与 newsuper。

知识点索引——超级用户

本文中把具有 SYSDBA 及 SYSOPER 权限的用户叫**超级用户**，超级用户是 Oracle 中的顶级用户。在数据库没有打开的情况下，Oracle 允许具有 SYSDBA 与 SYSOPER 权限的用户登录实例。SYSDBA 与 SYSOPER 的权力非常大。

SYSDBA 可完成的任务：

✓ 启动和关闭数据库（STARTUP 与 SHUTDOWN）

✓ 创建数据库（CREATE DATABASE）

✓ 删除数据库（DROP DATABASE）

✓ 创建 SPFILE（CREATE SPFILE）

✓ 改变数据库归档模式（ALTER DATABASE ARCHIVELOG）

✓ 恢复数据库（ALTER DATABASE RECOVER）

✓ 限制会话（RESTRICTED SESSION）

✓ 使用 ALTER DATABASE 打开和 MOUNT 数据库

✓ 使用 ALTER DATABASE 备份数据库

✓ 使用 ALTER DATABASE 改变数据库的字符集

SYSOPER 可完成的任务：

✓ 启动和关闭数据库（STARTUP 与 SHUTDOWN）

✓ 修改数据库的归档模式（ALTER DATABASE ARCHIVELOG）

✓ 恢复数据库（ALTER DATABASE RECOVER）

✓ 创建 SPFILE（CREATE SPFILE）

✓ 使用 ALTER DATABASE 打开和 MOUNT 数据库

✓ 使用 ALTER DATABASE 备份数据库

✓ 限制会话（RESTRICTED SESSION）

用户使用 CONNECT username/password AS SYSDBA 或者 CONNECT username/password AS SYSOPER 的方式登录数据库。用户登录到数据库以后，登录到指定的模式，而不是用户自己的模式。SYSDBA 登录的默认模式是 SYS；SYSOPER 登录的默认模式是 PUBLIC。



14.3.4 管理员的验证之——基于网络的验证

基于网络的验证，也叫强力验证（Strong Authentication），它集中控制 SYSDBA、SYSOPER 对数据库的访问。它把验证的工作交给目录服务器（Directory Server）来完成。目录服务器实现对 SYSDBA、SYSOPER 的验证。这使得验证的工作从数据库中脱离出来。目录服务器有 Oracle Internet Directory（OID），我们可以单独对这样的服务器进行管理。



14.4 权限管理

本节将对 Oracle 用户的权限进行介绍。



14.4.1 力量的象征——权限

权限是用户执行特殊的 SQL 语句和访问数据库对象的权力。有了权限，用户才能登录数据库、查询表等。有两种类型的权限：系统权限（System Privileges）和对象权限（Schema

Object Privileges）。

系统权限（System Privileges）

一个系统权限是执行一种特殊动作或者在一类对象上执行一种特殊动作的权力。也就是说，系统权限不只是针对单个数据库对象。Oracle 中有下面的系统权限，如表 14-2 所示。

表 14-2　系统权限说明

系统权限	系统权限说明
CREATE ANY TABLE	在任何用户模式下创建表的权限
CREATE TABLESPACE	创建表空间的权限
CREATE TRIGGER	在自己用户模式下创建触发器的权限
CREATE TYPE	在自己模式下创建类型的权限
CREATE USER	创建用户的权限
CREATE VIEW	在自己用户模式下创建视图的权限
CREATE ANY CLUSTER	在任何用户模式下创建聚簇的权限
CREATE ANY DIRECTORY	创建任何目录的权限
CREATE ANY INDEX	在任何模式下的表上创建索引的权限
CREATE ANY LIBRARY	在任何用户模式下创建库（LIBRARY）的权限
CREATE ANY PROCEDURE	在任何用户模式下创建存储过程、函数和包的权限
CREATE ANY SEQUENCE	在任何用户模式下创建任何序列的权限
CREATE ANY SYNONYM	在任何用户模式下创建任何同义词的权限
CREATE ANY TRIGGER	创建任何触发器的权限
CREATE ANY TYPE	在任何模式下创建类型的权限
CREATE ANY VIEW	创建任意视图的权限
CREATE CLUSTER	在自己用户模式下创建聚簇的权限
CREATE DATABASE LINK	在自己用户模式下创建专用数据库链接的权限
CREATE LIBRARY	在自己用户模式下创建库（LIBRARY）的权限
CREATE PROCEDURE	在自己用户模式下创建存储过程、函数和包的权限
CREATE PROFILE	创建资源限制文件的权限
CREATE PUBLIC DATABASE LINK	创建公共数据库链接的权限
CREATE PUBLIC SYNONYM	创建公共同义词的权限
CREATE ROLE	创建角色的权限
CREATE SEQUENCE	在自己用户模式下创建序列的权限
CREATE SESSION	连接到数据库的权限
CREATE SYNONYM	在自己用户模式下创建同义词的权限
CREATE TABLE	在自己用户模式下创建表的权限

续表

系统权限	系统权限说明
ALTER ANY CLUSTER	改变数据库中任何聚簇的权限
ALTER ANY INDEX	改变任何索引的权限
ALTER ANY PROCEDURE	改变任何用户模式下的存储过程、函数或包的权限
ALTER ANY ROLE	改变数据库中的任何一个角色的权限
ALTER ANY SEQUENCE	在任何用户模式下改变任何序列的权限
ALTER ANY TABLE	在任何用户模式下改变任何表的权限
ALTER ANY TRIGGER	修改任何触发器的权限
ALTER ANY TYPE	修改任何类型的权限
ALTER DATABASE	改变数据库的权限
ALTER PROFILE	改变资源限制文件的权限
ALTER RESOURCE COST	设置所有的用户会话中使用资源开销的权限
ALTER SESSION	执行 ALTER SESSION 命令的权限
ALTER SYSTEM	执行 ALTER SYSTEM 命令的权限
ALTER TABLESPACE	改变表空间的权限
ALTER USER	改变用户属性的权限
ANALYZE ANY	分析数据库中的任何表、聚簇或索引的权限
AUDIT ANY	审计数据库中的任何模式对象的权限
AUDIT SYSTEM	审计数据库的权限
COMMENT ANY TABLE	对任何用户模式下的任何表加注释的权限
DELETE ANY TABLE	删除任何用户模式下的任何表中的数据的权限
DROP ANY CLUSTER	删除数据库中任何一个聚簇的权限
DROP ANY INDEX	删除任何索引的权限
DROP ANY LIBRARY	删除任何用户模式下的库（LIBRARY）的权限
DROP ANY PROCEDURE	删除任何用户模式下的存储过程、函数或包的权限
DROP ANY ROLE	删除数据库中的任何一个角色的权限
DROP ANY SEQUENCE	在任何用户模式下删除任何序列的权限
DROP ANY SYNONYM	在任何用户模式下删除任何同义词的权限
DROP ANY TABLE	删除或截断任何用户模式下的任何表的权限
DROP ANY TRIGGER	删除任何用户模式下的任何触发器的权限
DROP ANY TYPE	删除任何类型的权限
DROP ANY VIEW	删除任意用户模式下的任意视图的权限
DROP PROFILE	删除资源限制文件的权限
DROP PUBLIC DATABASE LINK	删除公共数据库链接的权限
DROP PUBLIC SYNONYM	删除公共同义词的权限

续表

系统权限	系统权限说明
DROP TABLESPACE	删除表空间的权限
DROP USER	删除用户的权限
EXECUTE ANY PROCEDURE	执行任何过程或函数的权限
EXECUTE ANY TYPE	引用任何类型的权限
FORCE ANY TRANSACTION	强迫提交或者回滚本地数据库中悬而未决的任何分布式事务的权限
FORCE TRANSACTION	强迫提交或者回滚本地数据库中悬而未决的自身分布式事务的权限
GRANT ANY PRIVILEGE	授予任何系统特权的权限
GRANT ANY ROLE	授权数据库中的任何一个角色的权限
INSERT ANY TABLE	插入数据到任何用户模式下的任何表的权限
LOCK ANY TABLE	锁定任何用户模式下的任何表的权限
MANAGE TABLESPACE	使表空间离线或使表空间联机的权限
SELECT ANY SEQUENCE	在任何用户模式下查询任何序列的权限
SELECT ANY TABLE	对任何用户模式下的任何表进行查询的权限
SYSDBA 与 SYSOPER	具有这两个权限的用户可以执行数据库的启动与关闭
UNLIMITED TABLESPACE	使用户可以无限制使用任何表空间的权限
UPDATE ANY TABLE	修改任何用户模式下的任何表的数据的权限
CREATE ANY TABLE	在任何用户模式下创建表的权限
CREATE TABLESPACE	创建表空间的权限
CREATE TRIGGER	在自己用户模式下创建触发器的权限
CREATE TYPE	在自己模式下创建类型的权限
CREATE USER	创建用户的权限
CREATE VIEW	在自己用户模式下创建视图的权限
CREATE ANY CLUSTER	在任何用户模式下创建聚簇的权限
CREATE ANY DIRECTORY	创建任何目录的权限
CREATE ANY INDEX	在任何模式下的表上创建索引的权限
CREATE ANY LIBRARY	在任何用户模式下创建库（LIBRARY）的权限
CREATE ANY PROCEDURE	在任何用户模式下创建存储过程、函数和包的权限
CREATE ANY SEQUENCE	在任何用户模式下创建任何序列的权限
CREATE ANY SYNONYM	在任何用户模式下创建任何同义词的权限
CREATE ANY TRIGGER	创建任何触发器的权限
CREATE ANY TYPE	在任何模式下创建类型的权限
CREATE ANY VIEW	创建任意视图的权限

续表

系统权限	系统权限说明
CREATE CLUSTER	在自己的用户模式下创建聚簇的权限
CREATE DATABASE LINK	在自己用户模式下创建专用数据库链接的权限
CREATE LIBRARY	在自己用户模式下创建库（LIBRARY）的权限
CREATE PROCEDURE	在自己用户模式下创建存储过程、函数和包的权限
CREATE PROFILE	创建资源限制文件的权限
CREATE PUBLIC DATABASE LINK	创建公共数据库链接的权限
CREATE PUBLIC SYNONYM	创建公共同义词的权限
CREATE ROLE	创建角色的权限
CREATE SEQUENCE	在自己用户模式下创建序列的权限
CREATE SESSION	连接到数据库的权限
CREATE SYNONYM	在自己用户模式下创建同义词的权限
CREATE TABLE	在自己用户模式下创建表的权限
ALTER ANY CLUSTER	改变数据库中任何聚簇的权限
ALTER ANY INDEX	改变任何索引的权限
ALTER ANY PROCEDURE	改变任何用户模式下的存储过程、函数或包的权限
ALTER ANY ROLE	改变数据库中的任何一个角色的权限
ALTER ANY SEQUENCE	在任何用户模式下改变任何序列的权限
ALTER ANY TABLE	改变任何用户模式下的任何表的权限
ALTER ANY TRIGGER	修改任何触发器的权限
ALTER ANY TYPE	修改任何类型的权限
ALTER DATABASE	改变数据库的权限
ALTER PROFILE	改变资源限制文件的权限
ALTER RESOURCE COST	设置所有的用户会话中使用资源开销的权限
ALTER SESSION	执行 ALTER SESSION 命令的权限
ALTER SYSTEM	执行 ALTER SYSTEM 命令的权限
ALTER TABLESPACE	改变表空间的权限
ALTER USER	改变用户属性的权限
ANALYZE ANY	分析数据库中的任何表、聚簇或索引的权限
AUDIT ANY	审计数据库中的任何模式对象的权限
AUDIT SYSTEM	审计数据库的权限
COMMENT ANY TABLE	对任何用户模式下的任何表加注释的权限
DELETE ANY TABLE	删除任何用户模式下任何表中数据的权限
DROP ANY CLUSTER	删除数据库中任何一个聚簇的权限
DROP ANY INDEX	删除任何索引的权限

续表

系统权限	系统权限说明
DROP ANY LIBRARY	删除任何用户模式下库（LIBRARY）的权限
DROP ANY PROCEDURE	删除任何用户模式下的存储过程、函数或包的权限
DROP ANY ROLE	删除数据库中的任何一个角色的权限
DROP ANY SEQUENCE	在任何用户模式下删除任何序列的权限
DROP ANY SYNONYM	在任何用户模式下删除任何同义词的权限
DROP ANY TABLE	删除或截断任何用户模式下的任何表的权限
DROP ANY TRIGGER	删除任何用户模式下的任何触发器的权限
DROP ANY TYPE	删除任何类型的权限
DROP ANY VIEW	删除任意用户模式下的任意视图的权限
DROP PROFILE	删除资源限制文件的权限
DROP PUBLIC DATABASE LINK	删除公共数据库链接的权限
DROP PUBLIC SYNONYM	删除公共同义词的权限
DROP TABLESPACE	删除表空间的权限
DROP USER	删除用户的权限
EXECUTE ANY PROCEDURE	执行任何过程或函数的权限
EXECUTE ANY TYPE	引用任何类型的权限
FORCE ANY TRANSACTION	强迫提交或者回滚本地数据库中悬而未决的任何分布式事务的权限
FORCE TRANSACTION	强迫提交或者回滚本地数据库中悬而未决的自身分布式事务的权限
GRANT ANY PRIVILEGE	授予任何系统特权的权限
GRANT ANY ROLE	授权数据库中的任何一个角色的权限
INSERT ANY TABLE	插入数据到任何用户模式下的任何表的权限
LOCK ANY TABLE	锁定任何用户模式下的任何表的权限
MANAGE TABLESPACE	使表空间离线或使表空间联机的权限
SELECT ANY SEQUENCE	在任何用户模式下查询任何序列的权限
SELECT ANY TABLE	对任何用户模式下的任何表进行查询的权限
SYSDBA 与 SYSOPER	具有这两个权限的用户可以执行数据库的启动与关闭
UNLIMITED TABLESPACE	使用户可以无限制使用任何表空间的权限
UPDATE ANY TABLE	修改任何用户模式下任何表数据的权限

对象权限（Schema Object Privileges）

对象权限是在指定模式对象（表、视图、序列、过程、函数或包）上执行某种操作的权力。对象权限针对特定的对象，它分成下面几类。

- OLAP 权限（OLAP PRIVILEGES）
- 物化视图权限（ MATERIALIZED VIEW PRIVILEGES）

- 闪回权限（FLASHBACK DATA ARCHIVE PRIVILEGE）
- 库权限（LIBRARY PRIVILEGE）
- 目录权限（DIRECTORY PRIVILEGES）
- 表权限（Table Object Privileges）
- 存储过程权限（Procedure Privileges）
- 视图权限（View Privileges）
- 序列权限（Sequences Privileges）
- 函数和包的权限（Functions and Packages Privileges）
- 类型权限（Type Privileges）
- ……

表权限有：

- INSERT 允许获得授权的用户往指定的表中插入数据
- UPDATE 允许获得授权的用户修改指定的表中的数据
- DELETE 允许获得授权的用户删除指定的表中的数据
- SELECT 允许获得授权的用户查询指定的表



14.4.2 搞不清楚的问题——角色与权限

学习数据库时，最容易混淆“角色”和“权限”的概念。一个角色（Role）是一组相关权限的集合。角色其实也是权限，只是它结合了多个权限。角色方便了权限的管理，如果要给某一类用户授予多个权限，我们可以创建一个角色（这个角色包含了多个权限），当我们给用户授权时，只需要把角色授予用户即可，不需要对该用户多次授权。

如图 14-1 所示，角色 KK 由权限 CREATE TABLE、DROP ANY VIEW 和 DROP USER 组成。

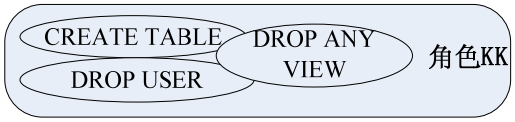


图 14-1　角色和权限的关系

数据库中有许多预定义的角色，如：

- CONNECT 角色包含下面的权限：CREATE SESSION、CREATE TABLE、CREATE VIEW、CREATE SYNONYM、CREATE SEQUENCE、CREATE DATABASE LINK、CREATE CLUSTER、ALTER SESSION。
- RESOURCE 角色包含下面的权限：CREATE TABLE、CREATE PROCEDURE、CREATE SEQUENCE、CREATE TRIGGER、CREATE TYPE、CREATE CLUSTER、CREATE INDEXTYPE、CREATE OPERATOR。
- DBA 角色包含大部分系统管理权限。

- SELECT_CATALOG_ROLE 表示查询数据字典的权限。
- EXECUTE_CATALOG_ROLE 表示从数据字典中执行部分过程和函数的权限。
- ……



14.4.3 创建角色

■ 创建普通角色 role1

```
CREATE ROLE role1;
```

■ 创建口令验证的角色 role2

```
CREATE ROLE role2
IDENTIFIED BY zero;
```

■ 创建全局角色（Global Role）role3

```
CREATE ROLE role3 IDENTIFIED GLOBALLY;
```

■ 创建外部角色（External Role）role4

```
CREATE ROLE role4 IDENTIFIED EXTERNALLY;
```



14.4.4 授予角色权限

■ 授予角色系统权限

```
GRANT CREATE VIEW TO role1;
```

■ 授予角色对象权限

```
GRANT INSERT ON tvb TO role1;
```

当然，也可以把角色授权给另外一个角色。



14.4.5 删除角色

删除角色 r1。

```
DROP ROLE r1;
```



14.4.6 权力演绎

STEP 01 一个漆黑的夜晚，用户 rawman 诞生了。

```
C:\Documents and Settings\Administrator>sqlplus /nolog
SQL*Plus: Release 11.1.0.6.0 - Production on 星期三 8月 26 08:47:06 200
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
```

```
SQL> connect sys as sysdba
输入口令:
SQL> CREATE USER rawman IDENTIFIED BY zero;
```

STEP 02 rawman 来到这个世界，一贫如洗，甚至连登录数据库的权力都没有。

```
C:\Documents and Settings\Administrator>sqlplus /nolog（在另外一个窗口启动SQL*Plus）
SQL> connect rawman
输入口令:
ERROR:
ORA-01045: user RAWMAN lacks CREATE SESSION privilege; logon denie
```

STEP 03 数据库管理员授予 rawman 登录数据库的权力，这是 rawman 获得的第一份权力，也是 rawman 获得的第一项系统权限。

```
SQL> GRANT CREATE SESSION TO rawman;
授权成功。
```

成功了！rawman 终于可以登录数据库了！

```
SQL> connect rawman
输入口令:
已连接。
```

STEP 04 数据库管理员授予 rawman 查询用户 bc 的表 t1 的权限。这是 rawman 获得的第一项对象权限。

```
SQL> GRANT SELECT ON bc.t1 TO rawman;
授权成功。
```

现在，rawman 可以查询别人的表，rawman 尝到了权力的甜头。

STEP 05 数据库管理员授予 rawman 创建表的权限（CREATE TABLE），rawman 可以建立自己的表了。

```
SQL> GRANT CREATE TABLE TO rawman;
授权成功。
```

STEP 06 数据库管理员授予 rawman 创建任何索引的权限。

```
GRANT CREATE ANY INDEX TO rawman WITH ADMIN OPTION;
授权成功。
```

令 rawman 感动的是，他居然可以把这个权限授予别人（bc）（因为选项 WITH ADMIN OPTION 选项）。

```
SQL> GRANT CREATE ANY INDEX TO bc;
授权成功。
```

STEP 07 不知足的 rawman 又要求更多的权力，于是数据库管理员把角色 role1 授予 rawman，rawman 继承了角色中的权限。这是 rawman 获得的第一个角色。

```
SQL> GRANT role1 TO rawman;
授权成功。
```

STEP 08 现在 rawman 变得更加强大。让我们看看 rawman 获得了哪些权限。

■ 查询 rawman 获得了哪些对象权限

```
SQL> SELECT TABLE_NAME,PRIVILEGE
  2  FROM DBA_TAB_PRIVS
  3  WHERE grantee='RAWMAN';
```

SQL 语句的执行结果如下所示：

```
TABLE_NAME                     PRIVILEGE
------------------------------ ----------------------------------------
T1                             SELECT
```

■ 查询 rawman 获得了哪些系统权限

```
SQL> SELECT *
  2  FROM DBA_SYS_PRIVS
  3  WHERE grantee='RAWMAN';
```

SQL 语句的执行结果如下所示：

```
GRANTEE          PRIVILEGE              ADM
------------------------------ ----------------------------------------
RAWMAN           CREATE TABLE           NO
RAWMAN           CREATE ANY INDEX       YES
RAWMAN           CREATE SESSION         NO
```

从 ADM= YES 可以看出，RAWMAN 能够把获得的权限（CREATE ANY INDEX）授予其他用户。

■ 查询 rawman 被授予哪些角色

```
SQL> SELECT GRANTED_ROLE
  2  FROM DBA_ROLE_PRIVS
  3  WHERE grantee='RAWMAN';
```

SQL 语句的执行结果如下所示：

```
GRANTED_ROLE
------------------------------
ROLE1
SQL>
```

后来，rawman 又要求更多更大的权力……

STEP 09 正所谓知足者常乐，rawman 却不是那样的人，rawman 的贪婪惹怒了数据库管理员，一怒之下，数据库管理员下定决心灭掉 rawman。

■ 数据库管理员收回授予 rawman 的系统权限

```
SQL> REVOKE CREATE SESSION FROM rawman;
```

```
撤销成功。

SQL> REVOKE CREATE TABLE FROM rawman;
撤销成功。

SQL> REVOKE CREATE ANY INDEX FROM rawman;
撤销成功。
```

■ 数据库管理员收回授予 rawman 的对象权限

```
SQL> REVOKE SELECT ON bc.t1 FROM rawman;
撤销成功。
```

■ 数据库管理员收回授予 rawman 的角色

```
SQL> REVOKE role1 FROM rawman;
撤销成功。
```

后来，rawman 又成为一个一贫如洗的人……



本章总结

本章主要介绍用户和权限。我们应该学会创建用户并给用户授权。学习本章时，你需要区分权限和角色的关系。关于用户概要文件（User Profiles），我们也许只需要了解一下即可，在实际生产环境中，用得其实并不多。有关用户的验证方式，还涉及数据库以外的知识，不能深究，我们只需学会使用足矣！

第 15 章　Oracle 数据泵（Oracle Data Pump）

在信息化时代，数据从一种载体被转移到另一种载体，从一种形式变换到另一种形式，需要从一个平台移动到另一个平台。数据的存储格式有文本文件、Excel、Access 等。我们常常需要把数据从文本文件导入到 Oracle，或者把 Oracle 中的数据移动到 SQL Server，或者把 Excel 的数据库装载到 Oracle。各个数据库厂商几乎都提供了数据迁移的工具，第三方厂商也提供了更加便捷的数据迁移工具。

本书只重点介绍 Oracle 数据库之间的数据迁移。Oracle 的数据迁移工具有 SQL*Loader，它被用于把外部数据装载到 Oracle 数据库中。从 Oracle 诞生的那天起，Oracle 从未停止过对速度的追求，Oracle 11g 推出了全新的数据迁移工具——Oracle 数据泵（Oracle Data Pump），它用于 Oracle 数据库之间的数据迁移。Oracle 数据泵可以实现数据的高速移动！数据泵是基于数据库服务器端的，它运行在数据库内部，而不是一个独立运行的客户端应用程序。

Oracle 以前的导入导出工具正渐渐地淡出我们的视线，推荐大家使用 Oracle 最新的导入导出工具——Oracle 数据泵！

15.1　Oracle 数据泵的组件

Oracle 数据泵由三部分组成：命令行客户端（expdp 和 impdp）、数据泵 API（DBMS_DATAPUMP）、元数据 API（DBMS_METADATA）。

如图 15-1 所示，expdp 和 impdp 调用数据泵 API 和元数据 API 共同完成数据的迁移。数据泵 API 执行表中数据的导入导出；元数据 API 完成元数据的装载和卸载。元数据（Metadata）是数据库的结构信息（Structural Information）。

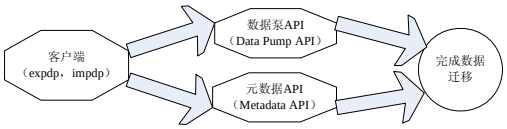


图 15-1　数据泵的组成



15.2 数据泵移动数据的方式

数据泵使用 4 种方式移动数据：数据文件复制（Data File Copying）、直接路径（Direct Path）、外部表（External Tables）、网络链接导入（Network Link Import）。

数据文件复制（Data File Copying）

这是最快的一种数据移动方法。它并不是从数据库中抽取数据，而只是把数据文件从一台数据库移动到另外一台数据库。Data Pump Export 只是把元数据（Metadata）导入到 DUMP 文件中。

直接路径（Direct Path）

直接路径是仅次于数据文件复制的数据移动方法，它绕过了 SQL 层，直接把行导出到 DUMP 文件中，或者直接把行导入数据库中。但是，直接路径有一些限制。如果这些限制存在，可能我们只能使用外部表方式。

外部表（External Tables）

外部表机制将创建一个外部表，这个外部表对应一个 DUMP 文件。外部表机制生成的 DUMP 文件的格式和普通导出方式生成的 DUMP 文件的格式相同。在导出数据时，使用外部表机制；在导入数据时，可以使用外部表机制（也可以使用其他方式）。外部表方式可以和其他方式联合使用，如导出数据时，使用直接路径；导入数据时，使用外部表机制。

网络链接导入（Network Link Import）

如果在导入时，指定参数 NETWORK_LINK（该参数用于指定网络连接），将使用“网络链接导入”方式，导入不会涉及 DUMP 文件，而是使用 INSERT SELECT 语句完成导入。SELECT 语句从远程数据库中检索数据，INSERT 语句完成往目标数据库中插入数据的任务。



15.3 数据泵的工作原理

主表（Master Table）

主表其实就是数据库中的一个表，使用它来保存整个导入导出过程的信息。这些信息包括数据库对象在 DUMP 文件中的位置、Job 状态等。Oracle 使用该表跟踪导入导出的过程。主表在进行当前导出、导入操作的用户模式下被创建。因此，该用户必须具有 CREATE TABLE 权限，且必须有足够的表空间。主表的名字和创建它的 Job 名字相同。导出的时候，主表被创建，并在完成的时候写到 DUMP 文件中。在启动导入的时候，主表被从 DUMP 文件集中载入到数据库中。

主进程（Master Process）

每一个导出导入的操作都会产生一个主进程， 主进程控制整个 Job，包括和客户端通信、控制**工作进程**（Worker Process）、执行日志操作。

工作进程（Worker Process）

真正执行卸载和装载元数据和表数据的进程是工作进程。工作进程的数量和 PARALLEL 指定的数值相等。工作进程并行工作。PARALLEL 是多少，就会分裂多少个工作进程。



15.4 目录问题

数据泵是基于服务器端的，当我们指定 DUMP 文件的目录时，我们说提及目录的也是基于服务器端的目录。这里的目录与客户端没有关系，只与服务器端有关。出于安全性的因素，Oracle 使用目录对象（Directory Object）来管理 DUMP 文件（Dump Files）、日志文件（Log Files）和 SQL 文件（SQL Files）。特别要注意的是，我们这里说的目录对象是一个数据库对象。必须要实际存在一个操作系统目录与之对应，并且 Oracle 要有该目录上的读写权限。

创建目录的过程如下：

STEP 01 在操作系统中，创建目录。

本例中创建目录 d:\dump

STEP 02 使 Oracle 对该目录有读写权限。

本例使 Oracle 对目录 d:\dump 有读写权限

STEP 03 登录数据库。

STEP 04 创建目录对象。

```
SQL> CREATE DIRECTORY dmp_dir AS 'd:\dump';
```

本例创建目录对象 dmp_dir。

STEP 05 使用户对目录对象有读写权限。

```
SQL> GRANT READ, WRITE ON DIRECTORY dmp_dir TO test;
```

本例中，授予用户 test 对目录 dmp_dir 具有读写权限。



15.5 导入导出过程的监控

通过查询视图，可了解导入导出的执行过程。从视图 DBA_DATAPUMP_JOBS（USER_DATAPUMP_JOBS）知道有多少个工作进程正在运行。DBA_DATAPUMP_SESSIONS 可以知道 JOB 的状态。V$SESSION_LONGOPS 可以帮助我们获得其他有用信

息来预测完成作业花费的时间。

15.6 数据泵导出工具（Data Pump Export）

数据泵导出工具(Data Pump Export)用于把数据库中的数据(包括元数据,即 Metadata)导成一系列的操作系统文件，这些文件叫做一个 DUMP 文件集（Dump File Set)。一个 DUMP 文件集由一个或者多个操作系统文件组成，我们通常把这些文件叫“DUMP 文件”，DUMP 文件可以被移动到其他系统，可以被导入到其他 Oracle 数据库，但是 DUMP 文件只能被工具 Data Pump Import 执行导入。DUMP 文件包括表的数据、元数据和控制信息等。

15.6.1 Data Pump Export 的调用接口

使用命令 expdp 调用 Data Pump Export，有三种调用接口。

- 命令行接口（Command-Line Interface)：在命令行，指定 expdp 的大部分参数。
- 参数文件接口（Parameter File Interface)：把命令行的参数放到一个文件中，在调用 expdp 时，使用参数 PARFILE 指向该文件，我们把该文件叫做参数文件。你需要留意的是，这里的参数文件和我们以前提到的 SPFILE、PFILE 是两个概念。这里的参数文件只针对数据泵。
- 交互式接口(Interactive-Command Interface)，执行 expdp 命令后，会出现交互窗口，用户可以根据提示一步一步完成导出。初级用户可以使用这种方式，不过，很多功能在交互模式下不能使用。

15.6.2 导出模式（Data Pump Export Modes）

我们可以导出整个数据库，也可以导出一部分数据，根据导出的不同，Oracle 提供了不同的导出模式。

- 全库导出模式（Full Export Mode）
- 模式导出（Schema Mode)，又叫用户导出模式
- 表导出模式（Table Mode）
- 表空间导出模式（Tablespace Mode）
- 可移动表空间导出模式（Transportable Tablespace Mode）

1．全库导出模式（Full Export Mode）

全库导出模式（Full Export Mode）将导出整个数据库（如图 15-2 所示)，执行全库导出的用户必须被授予 EXP_FULL_DATABASE 角色。全库导出的例子如下：

```
expdp tuser DIRECTORY=dmp_dir DUMPFILE=fulldb.dmp FULL=y
```

- FULL=y 表示执行全库导出。
- expdp 是操作系统命令，它将调用 Data Pump Export。
- tuser 是本例中执行导出的用户。
- DIRECTORY=dmp_dir 表示目录名，特别需要注意的是，这里的目录名是一个数据库对象。
- DUMPFILE=fulldb.dmp 表示 DUMP 文件的文件名。

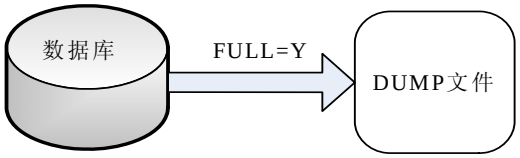


图 15-2　全库导出

2．模式导出（Schema Mode）

“模式导出”又叫“用户导出模式”，只导出一个或者多个用户的数据。若想导出其他用户模式的对象，执行导出的用户必须被授予 EXP_FULL_DATABASE 角色。

```
expdp TTM DIRECTORY= dmp_dir DUMPFILE=userdum.dmp SCHEMAS=cc,dd
```

本例中导出用户 cc 与 dd 的对象（如图 15-3 所示）。

- TTM 是本例中执行导出的用户。
- DIRECTORY 指定导出的目录。
- DUMPFILE 指定 DUMP 文件的名字。
- SCHEMAS 指定要被导出数据的用户。

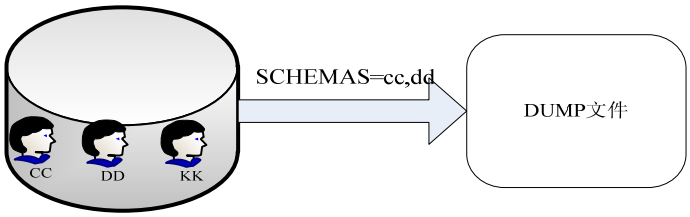


图 15-3　模式导出

3．表导出模式（Table Mode）

表导出模式（Table Mode）只导出参数 TABLES 指定的表。要想导出其他用户的表，执行导出的用户必须被授予 EXP_FULL_DATABASE 角色。表导出的例子（如图 15-4 所示）如下：

```
expdp SYSTEM DIRECTORY= dmp_dir DUMPFILE=DUMPtable.dmp
TABLES=tuser.infor,kk.sales:sales_P2
```

本例导出用户 tuser 的表 infor 的数据和用户 kk 的表 sales 的分区 sales_P2 的数据。

- expdp 是导出命令。

- SYSTEM 是执行导出的用户。
- DIRECTORY 指定导出目录。
- DUMPFILE 指定导出文件。
- TABLES 指定要被导出的表，可以同时指定多个表。

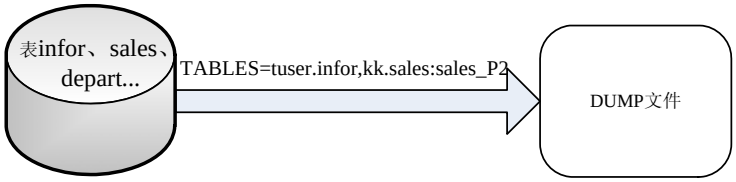


图 15-4　表导出模式

4．表空间导出模式（Tablespace Mode）

如图 15-5 所示，表空间导出模式（Tablespace Mode）导出一个或者多个表空间中的对象。只有在该表空间中的表才能被导出。如果表空间中的一个表被导出，则与其关联的对象也将被导出。表空间导出的例子如下：

```
expdp SYSTEM DIRECTORY=dmp_dir DUMPFILE=DUMPtbs.dmp
TABLESPACES=tbs1, tbs2
```

本例导出表空间 tbs1 及 tbs2 中的对象。

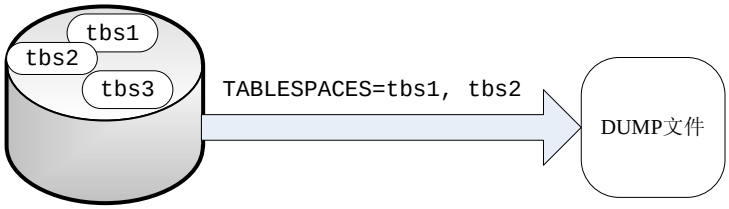


图 15-5　表空间导出模式

- expdp 是导出命令。
- SYSTEM 是执行导出的用户。
- DIRECTORY 指定导出目录。
- DUMPFILE 指定导出文件。
- TABLESPACES 指定要被导出的表空间，可以指定多个。

5．可移动表空间导出模式（Transportable Tablespace Mode）

可移动表空间导出模式（Transportable Tablespace Mode）只导出在表空间中表的元数据（Metadata），并不真正导出数据，数据依然存放在该表空间的数据文件中（如图 15-6 所示）。执行导入前，执行导出的用户需要把 DUMP 文件及数据文件复制到目标机器。可移动表空间导出的例子如下：

```
expdp SYSTEM DIRECTORY=dmp_dir DUMPFILE=DUMPtbs.dmp
```

```
TRANSPORT_TABLESPACES=tbs1
```

本例导出表空间 tbs1 中的元数据，存放在 DUMP 文件 DUMPtbs.dmp 中。

- expdp 是导出命令。
- SYSTEM 是执行导出的用户。
- DIRECTORY 指定导出目录。
- DUMPFILE 指定导出文件。
- TRANSPORT_TABLESPACES 指定要被移动的表空间，可以指定多个。
- LOGFILE 指定导出的日志文件的名字，日志文件记录导出的过程。

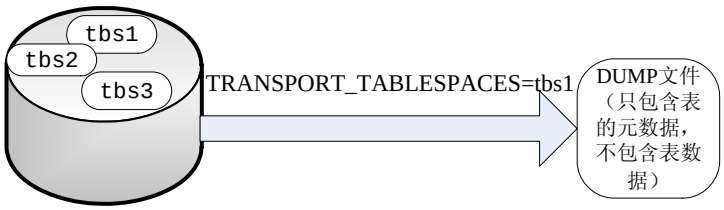


图 15-6　可移动表空间导出模式

15.6.3　导出参数说明

对于导出参数，有的在命令行接口（Command-Line Interface）有效，有的在交互式接口（Interactive-Command Interface）有效，有的在两个地方都有效。对于 expdp 参数，不必记住哪些参数在哪个接口有效，只需记住参数的用法足矣！使用的时候，只要查看 Oracle 的官方文档，便一目了然。

1．在命令行接口（Command-Line Interface）有效的参数

本节将介绍在命令行接口有效的参数。

📖 ATTACH

ATTACH [=[schema_name.]job_name]把会话和正在运行的导出任务（job）进行关联。

例：

```
expdp eus ATTACH=eus.export_job
```

其中，eus 是用户的名字，export_job 是任务的名字。

📖 COMPRESSION

COMPRESSION={ALL | DATA_ONLY | METADATA_ONLY | NONE}指定哪些内容在被写到 DUMP 文件之前要进行压缩。

- ALL 表示将压缩对象定义及其所有数据。
- DATA_ONLY 表示只压缩对象数据。
- METADATA_ONLY 表示只压缩对象定义的信息。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=eus_comp.dmp
COMPRESSION=METADATA_ONLY
```

📖 CONTENT

CONTENT={ALL | DATA_ONLY | METADATA_ONLY | NONE}指定哪些内容被导出：

- ALL 表示导出对象定义及其所有数据。
- DATA_ONLY 表示导出对象数据。
- METADATA_ONLY 表示导出对象定义。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=eus.dmp CONTENT=METADATA_ONL
```

📖 DIRECTORY

DIRECTORY=directory_object 指定目录对象。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=epm.dmp
```

📖 DUMPFILE

DUMPFILE=[directory_object:]file_name [, ...]
指定 DUMP 文件及其路径

例：

```
expdp eus SCHEMAS=eus DIRECTORY=dmp_dir DUMPFILE=dpump_dir2:exp1.dmp,
exp2%U.dmp
```

替换符（%U）表示可以自动产生多个 DUMP 文件。

📖 ENCRYPTION

ENCRYPTION = {ALL | DATA_ONLY | ENCRYPTED_COLUMNS_ONLY | METADATA_ONLY | NONE}指定哪些数据被写到 DUMP 文件之前要进行加密。

- ALL 表示对象定义及其所有数据。
- DATA_ONLY 表示对象数据。
- METADATA_ONLY 表示对象定义。
- ENCRYPTED_COLUMNS_ONLY 表示只加密写入到 DUMP 文件中的列。
- NONE 表示不加密。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=eus_enc.dmp JOB_NAME=enc1
ENCRYPTION=data_only ENCRYPTION_PASSWORD=cooboy
```

📖 ENCRYPTION_ALGORITHM

ENCRYPTION_ALGORITHM = { AES128 | AES192 | AES256 }指定加密算法。有三种加密算法。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=eus_enc.dmp
```

```
ENCRYPTION_PASSWORD=cooboy ENCRYPTION_ALGORITHM=AES128
```

📖 ENCRYPTION_MODE

ENCRYPTION_MODE = { DUAL | PASSWORD | TRANSPARENT }指定加密模式。

- PASSWORD 表示导出时需要设置口令，在导入时需要提供口令。
- TRANSPARENT 表示用 Oracle 加密钱夹（Encryption Wallet）创建 DUMP 文件。
- DUAL 表示创建的 DUMP 文件可以被前两种模式导入。

📖 ENCRYPTION_PASSWORD

ENCRYPTION_PASSWORD 指定加密口令，用于对 DUMP 文件进行加密。

例：

```
expdp eus TABLES=inff_encrypt DIRECTORY=dpump_dir
DUMPFILE= eus_comp ENCRYPTION=ENCRYPTED_COLUMNS_ONLY
ENCRYPTION_PASSWORD=9977
```

本例中的加密口令是 9977。

📖 ESTIMATE_ONLY

如果 ESTIMATE_ONLY=y 将评估一个 Job 将要消耗多少磁盘空间，并不真正执行导出。

例：

```
expdp eus ESTIMATE_ONLY=y SCHEMAS=EUS
```

📖 ESTIMATE

ESTIMATE={BLOCKS | STATISTICS}设置统计方法。

- BLOCKS 表示 Oracle 会按照目标对象所占用的数据块个数乘以数据块尺寸估算对象占用的空间。
- STATISTICS 表示将根据统计估算对象占用空间。

📖 EXCLUDE

EXCLUDE=object_type[:name_clause] [，...]该选项用于指定执行操作时要排除的对象类型或相关对象。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=eus_exclude.dmp EXCLUDE=VIEW,
FUNCTION
```


📖 FILESIZE

FILESIZE=integer[B | K | M | G]指定导出文件的最大尺寸，默认为 0（表示文件尺寸没有限制）。

📖 FLASHBACK_SCN

FLASHBACK_SCN=scn_value 指定导出特定 SCN（System Change Number）时刻的表数据。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=eus_scn.dmp FLASHBACK_SCN=254432
```

📖 FLASHBACK_TIME

指定导出特定时间点的表数据。

FLASHBACK_TIME="TO_TIMESTAMP(time-value)"

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=eus_scn.dmp FLASHBACK_TIME="TO_TIMESTAMP('8-08-2008 4:05:00', 'DD-MM-YYYY HH24:MI:SS')"
```

📖 FULL

FULL={y | n}

FULL 指定是否使用全库导出，FULL=Y 表示使用全库导出。

📖 HELP

HELP = {y | n}

HELP=Y 表示获取帮助。

📖 INCLUDE

INCLUDE = object_type[:name_clause] [, ...]指定被导出的对象类型和相关对象。

例：

```
INCLUDE=TABLE:"IN ('EPM', 'DEPARTMENTS')"
```

📖 JOB_NAME

JOB_NAME=jobname_string 用于指定导出作业名。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=exp_job.dmp JOB_NAME=exp_job
```

📖 LOGFILE

LOGFILE=[directory_object:]file_name 指定导出日志的名字及路径，默认值是 export.log。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE=eus.dmp LOGFILE=eus_export.log
```

📖 NETWORK_LINK

NETWORK_LINK=source_database_link 指定数据库链接（Database Link），如果要将远程数据库对象导出到本地实例的 DUMP 文件中，要指定该选项。expdp 将从 source_database_link 指向的数据库中检索数据。

例：

```
expdp eus DIRECTORY=dmp_dir NETWORK_LINK=sdblink
DUMPFILE=network_export.dmp LOGFILE=network_export.log
```

📖 NOLOGFILE

NOLOGFILE={y | n}

NOLOGFILE=y 表示不写日志文件。默认情况要写日志文件。

📖 PARALLEL

PARALLEL=integer 指定并行度。

📖 PARFILE

PARFILE=[directory_path]file_name 指定参数文件。参数文件用于存放参数。

参数文件使用的例子：

1）用文本编辑器创建参数文件 pkk。

```
DIRECTORY=dmp_dir
DUMPFILE=eus.dmp
TABLES=epm,jobs
```

2）执行导出，引用参数文件。

```
expdp eus PARFILE=pkk
```

📖 QUERY

QUERY = [schema.][table_name:] query_clause

QUERY 指定一个过滤条件，只有满足过滤条件的数据才被导出。

例：

```
QUERY=epm:"WHERE id > 10 AND salary < 999"
```

📖 REMAP_DATA

REMAP_DATA=[schema.]tablename.column_name:[schema.]pkg.function

这个功能在导入/导出操作时可以用来改变源数据，将原始数据替换成其他数据。将改变后的数据导入目标数据库。这个功能比较适合把生产环境的数据导入测试环境时使用。

📖 REUSE_DUMPFILES

REUSE_DUMPFILES={Y | N}指定是否覆盖已经存在的 DUMP 文件。

📖 SAMPLE

SAMPLE=[[schema_name.]table_name:]sample_percent 指定多少百分比的数据将被导出，取值从.000001 到 100。

例：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE= eus.dmp SAMPLE=20
```

📖 SCHEMAS

SCHEMAS=schema_name [, ...]导出指定模式下的对象。

📖 STATUS

STATUS=[integer]，单位是秒，表示每隔多长时间显示导出的信息。

例：

```
expdp eus DIRECTORY=dmp_dir SCHEMAS=eus STATUS=400
```

📖 TABLES

TABLES=[schema_name.]table_name[:partition_name] [, ...]表示导出指定的表。

📖 TABLESPACES

TABLESPACES=tablespace_name [, ...]表示导出指定表空间中的表。

📖 TRANSPORT_FULL_CHECK

TRANSPORT_FULL_CHECK={y | n}指定是否对要对被导出的表空间中的对象和表空间外的对象进行关联性检查。

📖 TRANSPORT_TABLESPACES

TRANSPORT_TABLESPACES=tablespace_name [, ...]指定可移动表空间，这些表空间中的对象的定义将被导出。只有可移动表空间导出模式（Transportable Tablespace Mode）才使用该参数。

📖 TRANSPORTABLE

TRANSPORTABLE = {ALWAYS | NEVER}和迁移表空间类似，TRANSPORTABLE 方式也不会导出真正的数据，只是导出对象的定义，唯一不同的是 TRANSPORTABLE 的对象是表，而不是整个表空间。导入前，需要将 DUMP 文件和数据文件发送到目标数据库服务器。其中，ALWAYS 表示使用可移动表特性；NEVER 不适用可移动表特性。

执行导出的命令如下：

```
expdp eus DIRECTORY=dmp_dir DUMPFILE= eus.dmp
TABLES= eus.depart_infor TRANSPORTABLE=always
```

执行导入的命令如下：

```
impdp SYSTEM directory=dir_output dumpfile= eus.dmp transport_datafiles=
'D:\U01.dbf'
```

📖 VERSION

VERSION={COMPATIBLE | LATEST | version_string}指定数据库对象的版本。通常情况下，不需要设置该参数，只有存在版本兼容性时才需要设置该参数。

2．在交互式接口（Interactive-Command Interface）有效的参数

下面我们将主要介绍在交互式接口有效的命令。

📖 ADD_FILE

ADD_FILE=[directory_object:]file_name [,...]表示增加 DUMP 文件到 DUMP 文件集中。

例：

```
Export> ADD_FILE=eus2.dmp, dpump_dir2:eus3.dmp
```

📖 CONTINUE_CLIENT

CONTINUE_CLIENT 命令将模式从交互式模式更改为记录模式（Logging Mode）。在记录模式下，状态信息不断在终端上显示。

例：

```
Export> CONTINUE_CLIENT
```

📖 EXIT_CLIENT

该命令退出导出，退到操作系统的命令窗口，但作业仍在运行。

例：

```
Export> EXIT_CLIENT
```

📖 FILESIZE

FILESIZE=number 设置 DUMP 文件大小，后续的导出将使用该设置。

例：

```
Export> FILESIZE=500M
```

📖 HELP

该命令能够获得帮助信息。

例：

```
Export> HELP
```

📖 KILL_JOB

该命令杀掉正在运行的作业。被杀掉的作业不能被重新启动。

例：

```
Export> KILL_JOB
```

📖 PARALLEL

该命令为当前的作业设置并行度。

例：

```
Export> PARALLEL=20
```

📖 START_JOB

该命令启动当前的作业。

例：

```
Export> START_JOB
```

📖 STATUS

STATUS[=integer] 单位是秒，表示每隔多长时间显示一次导出的状态信息。

例：

```
Export> STATUS=100
```

📖 STOP_JOB

STOP_JOB[=IMMEDIATE]表示停止作业，停止作业和杀掉作业不一样，停止的作业可以使用 START_JOB 命令重新启动。

15.7 数据泵导出工具（Data Pump Import）

如图 15-7 所示，数据泵导入工具（Data Pump Import）用于把 DUMP 文件的数据导入目标数据库。和 Data Pump Export 正好相反，Data Pump Import 是导入数据，而 Data Pump Export 是导出数据。



图 15-7　导出工具和导入工具的区别

导出和导入都是基于数据库服务器端的，因此都会用到目录对象。有关目录对象的信息，**请参考 15.4 节的内容**。

和 Data Pump Export 的调用类似，我们使用 impdp 命令调用 Data Pump Import，有三种调用接口。

- 命令行接口（Command-Line Interface）
- 参数文件接口（Parameter File Interface）
- 交互式接口（Interactive-Command Interface）

有关这几种接口的说明，**请参考 15.6.1 节的内容**。

和 Data Pump Export 类似，Data Pump Import 提供以下几种导入模式：

- 全库导入模式（Full Import Mode）
- 模式导入（Schema Mode），又叫用户导入模式
- 表导入模式（Table Mode）
- 表空间导入模式（Tablespace Mode）
- 可移动表空间导入模式（Transportable Tablespace Mode）

有关这几种导入模式的信息，**请参考 15.6.2 节的内容**。

对于 impdp，有的参数在命令行接口（Command-Line Interface）有效，有的参数在交互式接口（Interactive-Command Interface）有效，有的参数在两个地方都有效，接下来我们将分别进行介绍。

1．在命令行接口（Command-Line Interface）有效的参数

下面我们将主要介绍在命令接口有效的参数。

📖 ENCRYPTION_PASSWORD

ENCRYPTION_PASSWORD = *password* 表示如果在导出时指定了加密密码，在导入时必须提供加密密码，两个密码必须相同。

📖 NETWORK_LINK

NETWORK_LINK=source_database_link 表示通过 source_database_link 检索数据，并

把数据直接插入到当前连接的数据库（目标数据库）中。整个移动数据的过程与 DUMP 文件无关。

例：

```
impdp eus TABLES=emt DIRECTORY=dmp_dir
NETWORK_LINK=source_database_link
```

📖 REMAP_DATA

REMAP_DATA=[schema.]tablename.column_name:[schema.]pkg.function 表示对某列的数据进行映射，然后导入目标数据库。映射通过函数来实现，所谓映射就是对原数据进行改变，导入数据库中的值是改变后的值。

例：

```
impdp eus DIRECTORY=dmp_dir DUMPFILE=expschema.dmp
TABLES=eus.emt REMAP_DATA=eus.emt.first_name:eus.r_pkg.plusx
```

📖 REMAP_DATAFILE

REMAP_DATAFILE=source_datafile:target_datafile 该选项用于可移动表空间。当在不同的操作系统之间移动数据文件时，使用该选项。该选项用于将原数据文件名与目标数据文件名进行映射（也就是建立对应关系）。也可使用该参数重命名数据文件。

```
REMAP_DATAFILE='D:\dbdata\tbs6.f':'/db1/eusdata/payroll/tbs6.f'
```

总而言之，这个选项可以实现数据文件之间的映射。

📖 REMAP_SCHEMA

REMAP_SCHEMA=source_schema:target_schema 将一个用户（source_schema）的数据导入另外一个用户 target_schema。

例：

```
impdp system DIRECTORY=dmp_dir DUMPFILE=eus.dmp REMAP_SCHEMA=eus:tt
```

总而言之，这个选项可以实现模式（用户）之间的映射。

📖 REMAP_TABLE

REMAP_TABLE=[schema.]old_tablename[.partition]:new_tablename 该选项可以实现表之间的映射，也可以实现表的重命名。

例：

```
impdp eus DIRECTORY=dmp_dir DUMPFILE=expsdata.dmp
TABLES=eus.emt REMAP_TABLE=eus.emt:empt
```

📖 REMAP_TABLESPACE

REMAP_TABLESPACE=source_tablespace:target_tablespace 表示将源表空间的所有对象导入到目标表空间中。

例：

```
impdp eus REMAP_TABLESPACE=s_tbs:t_tbs DIRECTORY=dmp_dir
```

```
DUMPFILE=emt.dmp
```

🕮 REUSE_DATAFILES

REUSE_DATAFILES={y | n}指定导入时是否重用（覆盖）已经存在的数据文件。

🕮 SKIP_UNUSABLE_INDEXES

SKIP_UNUSABLE_INDEXES={y | n}指定如果表上有无用的索引，是否跳过该表，不进行装载。

🕮 SQLFILE

SQLFILE=[directory_object:]file_name 表示执行导入时，将执行一些 DDL 语句，是否将这些 DDL 语句写入一个文件中。这里指定该文件，通常把该文件叫做“SQL 文件”。

🕮 TABLE_EXISTS_ACTION

TABLE_EXISTS_ACTION={SKIP | APPEND | TRUNCATE | REPLACE}表示如果目标表已经存在，如何进行处理：

- SKIP 表示跳过目标表；
- APPEND 表示往已经存在的表中追加数据；
- TRUNCATE 表示清空已经存在的数据，然后执行导入；
- REPLACE 表示删除（DROP）已经存在的表，然后重建该表，再执行数据导入。

🕮 STREAMS_CONFIGURATION

STREAMS_CONFIGURATION={y | n}指定是否导入有关流的元数据（定义）。

🕮 TRANSFORM

TRANSFORM = transform_name:value[:object_type]该选项用于指定是否修改建立对象的 DDL 语句。

🕮 TRANSPORT_DATAFILES

TRANSPORT_DATAFILES=datafile_name 表示在可移动表空间模式（Transportable Tablespace Mode）下，指定要被导入的数据文件。

例：

```
TRANSPORT_DATAFILES='D:\tbs1.DBF'
```

🕮 **其他参数说明**

其他的 impdp 参数（如 ATTACH、CONTENT、DIRECTORY、DUMPFILE 等）与 expdp 的参数类似，这里就不再赘述！

2．在交互式接口（Interactive-Command Interface）有效的参数

无论是导入还是导出，在交互式接口使用的参数都类似，欲知详细信息，**请参考 15.6.3 节的内容**。



本章总结

本章主要介绍 Oracle 数据泵。Oracle 数据泵与 Oracle 11g 以前的导入导出工具不同，它不能与以前的导入导出工具兼容。如果你是一个初学者，建议学习 Oracle 数据泵，因为 Oracle 以前的导入导出工具正在被逐渐淘汰！学习本章的方法是要多实践，实践次数多了，你自然而然就会理解数据泵的相关概念。

第 16 章　学习任何数据库必学之——SQL 语句

本章将介绍数据库中各种类型的表，并重点介绍 SQL 语句。本章的亮点是，展示在开发过程中 SQL 语句的使用技巧！因为追求技巧性，所以本章耗费了笔者大量的时间，不过，如果读者愿意和笔者一起分享，那样的付出是值得的！



16.1　SQL 简述

SQL 即结构化查询语言（Structured Query Language），它是一种用于和关系数据库进行交互通信的语言，用于对数据库进行操作。最早使用在 IBM 公司开发的数据库系统中，后来，SQL 发展成为行业标准。SQL Server、Oracle、MySQL、DB2、PostgreSQL 等都支持 SQL，只是各个数据库厂商的支持有略微差异。

SQL 是高级的非过程化编程语言，它允许用户在高层数据结构上工作，而不要求用户指定数据的存取方法，也不需要用户了解数据的具体存放方式。

无论你学习任何数据库，都必须学习 SQL 语句，SQL 语句是一种标准，只要精通一个数据库厂商的 SQL 语句，也就差不多精通其他数据库厂商的 SQL 语句，这就是所谓的“学习任何数据库必学之——SQL 语句”。

SQL 语言细分成以下 6 种（不同的书籍有不同的分类方法，有的分得比较粗略，有的分得比较详细）。

- 数据定义语言（Data Definition Language，DDL）
- 数据操作语言（Data Manipulation Language，DML）
- 事务控制语言（Transaction Control Language，TCL）
- 会话控制语言（Session Control Language）
- 系统控制语言（System Control Language）
- 内置 SQL（Embedded SQL）

数据定义语言（Data Definition Language，DDL）

DDL 用于定义和改变模式对象（如表、视图……）的结构。其实就是改变对象的定义。

DDL 可以：

- 创建（CREATE）、改变（ALTER）、删除（DROP）模式对象及其他数据库结构（包括数据库本身及数据库用户）
- 重命名（RENAME）数据库对象
- 删除所有数据（TRUNCATE）
- 管理权限（GRANT、REVOKE）
- 设置审计（AUDIT、NOAUDIT）
- 增加注释（COMMENT）到数据字典

Oracle 会隐式提交 DDL 语句（即不需要用户输入 COMMIT 命令）。DDL 语句的例子如下所示。

示例 1：

```
CREATE TABLE uinfor
(NAME VARCHAR2 (20),
ID VARCHAR2 (30));
```

示例 2：

```
DROP TABLE uinfor;
```

示例 3：

```
GRANT SELECT ON uinfor TO rawman;
```

示例 4：

```
REVOKE DELETE ON uinfor FROM rawman;
```

数据操作语言（Data Manipulation Language，DML）

DML 用于查询或者操作数据库对象（主要指表）中的数据。特别注意的是，DML 只改变表中的数据，并不会改变表的结构（表结构的改变由 DDL 语句来完成）。DML 可以：

- 从表或者视图中检索数据（SELECT）。
- 往表或者视图中插入数据（INSERT）
- 修改列的值（UPDATE）
- 有条件地往表或视图中插入或者更新数据（MERGE）
- 从表或者视图中删除数据（DELETE）
- 查看 SQL 语句的执行计划（EXPLAIN PLAN）
- 锁住表及用户

事务控制语言（Transaction Control Language）

事务控制语言主要用于事务控制，包括提交（COMMIT）、回滚（OLLBACK）、设置回滚点（SAVEPOINT）、设置事务（SET TRANSACTION）。

会话控制语言（Session Control Language）

会话控制语言用于控制会话（ALTER SESSION）。

系统控制语言（System Control Language）

系统控制语言用于修改数据库实例的属性（ALTER SYSTEM）（如修改共享服务器进程的最大数量）。

内置 SQL（Embedded SQL）

内置 SQL 可以：

- 定义、分配、释放游标（DECLARE CURSOR、OPEN、CLOSE）
- 指定数据库或者连接数据库（DECLARE DATABASE、CONNECT）
- 变量设置（DECLARE STATEMENT）
- 描述对象（DESCRIBE）
- 错误或者警告处理（WHENEVER）
- 解析或者运行 SQL 语句（PREPARE、EXECUTE、EXECUTE IMMEDIATE）
- 从数据库中获得数据（FETCH）

16.2 事务（Transaction）

如图 16-1 所示，一个事务（Transaction）由一个或者多个 SQL 语句组成，这些 SQL 语句被看做一个**整体**，它们要么被**一起**提交（COMMIT），要么被**一起**回滚（ROLLBACK），这叫事务的原子性（Atomicity）。一个事务以第一个 SQL 语句开始，以 COMMIT 或者 ROLLBACK 结束（DDL 语句隐含提交 SQL 语句，不需要我们输入 COMMIT 命令），本文主要讲述显式提交（“显式提交”就是需要用户输入 COMMIT 命令）。

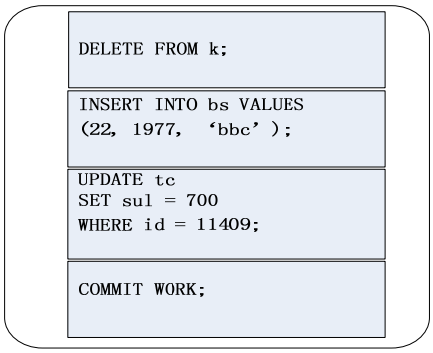


图 16-1 事务

16.2.1 提交（COMMIT）

所谓“提交（COMMIT）”，就是使对数据库的改变永久化。提交以前，对数据库的改变只是写到回滚段中，并未真正写到数据库中。使用 COMMIT 语句执行提交。语句

COMMIT WORK 和语句 COMMIT 等价。SQL 语句的提交方式如图 16-2 所示。

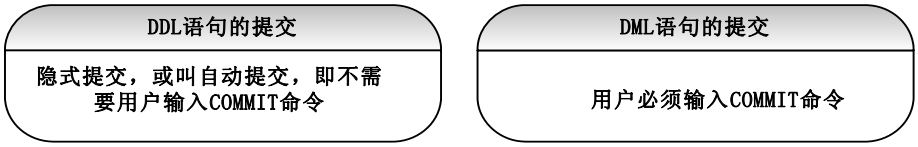


图 16-2　COMMIT

16.2.2　回滚（ROLLBACK）

所谓“回滚（ROLLBACK）”，就是撤销“未提交”的事务对数据库的改变，这里特别强调“未提交”。已经提交（COMMIT）的事务不能回滚。

小提示：DDL 语句隐式提交，不能回滚。

16.2.3　事务命名（SET TRANSACTION NAME）

我们可以使用 SET TRANSACTION NAME 命令给事务命名。事务命名的好处在于方便事务的监控。事务命名的例子如下所示。

STEP 01 命名事务。

```
SQL> SET TRANSACTION NAME 'INSERT INTO table tra';
事务处理集。
```

本例中，'INSERT INTO table tra'是事务的名字。

STEP 02 往表中插入数据。

```
SQL> INSERT INTO tra
  2  VALUES(2,'kk');
已创建 1 行。
```

STEP 03 查看事务是否存在。

```
SQL> SELECT name FROM v$transaction;
NAME
----------------------------------------------------------
INSERT INTO table tra
```

从查询结果可以看出，事务已经存在。从视图 v$transaction 中，我们还可以获得事务的其他信息。

STEP 04 提交事务。

```
SQL> COMMIT;
提交完成。
```

STEP 05 提交完成后，查看事务的存在性。

```
SQL> SELECT name FROM v$transaction;
```

```
未选定行
```

提交完成后，事务不复存在。

16.2.4 事务读一致性(Transaction-level Read Consistency)

使用 SET TRANSACTION READ ONLY 命令设置事务读一致性。所谓“事务读一致性”是指在该事务中的所有查询语句，只能看到 COMMIT 以前的数据（也就是 SET TRANSACTION READ ONLY 之前的数据，不能看到 SET TRANSACTION READ ONLY 之后改变的数据）。事务读一致性的例子如下所示。

STEP 01 在会话一（或叫窗口一）执行下面的操作。

ⓐ 设置事务读一致。

```
SQL> SET TRANSACTION READ ONLY;
    事务处理集。
```

ⓑ 查询表 tra 中的数据。

```
SQL> SELECT * FROM tra;
     A B
---------- ----
     2 kk
```

表 tra 中只有一条记录。

STEP 02 在会话二（或叫窗口二）执行下面的操作。

ⓐ 插入一行数据。

```
SQL> INSERT INTO tra VALUES(3,'bb');
已创建 1 行。
```

ⓑ 提交数据。

```
SQL> COMMIT;
提交完成。
```

STEP 03 在会话一（或叫窗口一）中查看表 tra 中的数据。

```
SQL> SELECT * FROM tra;
     A B
---------- ----
    2 kk
```

尽管会话二往表 tra 中添加并提交了一条新的记录，但是会话一看不到新添加的记录，因为会话一只能看到 SET TRANSACTION READ ONLY 之前的数据。

STEP 04 在会话一（或叫窗口一）执行下面的操作。

ⓐ 结束只读事务。

```
SQL> COMMIT;
提交完成。
```

ⓑ 第三次查看表 tra 中的数据。

```
SQL> SELECT * FROM tra;
                         A B
                         ---------- ----
                         2 kk
                         3 bb
```

因为只读事务已经结束，所以会话一可以看到其他用户提交的数据。

只读事务内只能执行下面的 SQL 语句，否则将会报告错误“ORA-01456：不能在 READ ONLY 事务处理中执行插入/删除/更新操作”：

- 不带 FOR UPDATE 的 SELECT 语句
- 锁表（LOCK TABLE）
- 设置角色（SET ROLE）
- 修改会话（ALTER SESSION）
- 修改实例设置（ALTER SYSTEM）

16.2.5 回滚点（SAVEPOINT）

回滚点是一个事务中的标记，它把一个事务分成几段。其目的是允许用户回滚事务的一部分，而不是整个事务。回滚点使用 SAVEPOINT 命令进行设置。回滚点的例子如下所示。

STEP 01 更新表 eps。

```
UPDATE eps
SET ID = 53221
WHERE name = 'BEN';
```

STEP 02 设置回滚点 sp1。

```
SAVEPOINT sp1;
```

STEP 03 再次更新表 eps。

```
UPDATE eps
SET ID = 77225
WHERE name = 'XIAO MING';
```

STEP 04 设置回滚点 sp2。

```
SAVEPOINT sp2;
```

STEP 05 回滚到回滚点 sp1。

```
ROLLBACK TO SAVEPOINT sp1;
```

也就是 sp1 以后的工作都被回滚。

16.2.6 自治事务（Autonomous Transactions）

自治事务机制允许用户在事务中建立一个新的事务，也就是事务中嵌套一个新事务（事务中存在另外一个事务），我们把这个新事务叫自治事务（Autonomous Transaction，AT），把原来的事务叫主事务（Main Transaction，MT，或者叫“父事务”）。自治事务的提交和回滚不会影响到父事务的状态。自治事务由主事务调用，但是独立于主事务，在自治事务被调用执行时，主事务被挂起，在自治事务内部，可以执行一系列的 DML 语句，并且可以提交或回滚。自治事务 AT 的提交或回滚并不影响主事务的执行。当自治事务执行结束后，主事务获得控制权，继续执行其他 SQL 代码。使用 PRAGMA AUTONOMOUS_TRANSACTION 指定自治事务。

16.3 数据类型（Oracle Data Type）

Oracle 的数据类型有内置的数据类型（Built-in Database Types，或者叫标准数据类型）和用户自定义的数据类型两种。当创建表时，需要指定列的数据类型，Oracle 内置的数据类型有：

- CHAR 存放固定长度的字符串。
- VARCHAR2 存放变长的字符串，最大长度是 4000 字节。
- VARCHAR 是 VARCHAR2 的同义词，推荐使用 VARCHAR2。
- NVARCHAR2 存放变长的 Unicode 字符数据，最大长度是 4000 字节。
- NVARCHAR 存放定长的 Unicode 字符数据，最大长度是 2000 字节。
- LONG 存放变长的字符数据，最大长度是 2GB。保留这种数据类型只是为了向后兼容，这种数据类型将被 LOB 所替代。建议不要使用这种数据类型。
- NUMBER 存放正负整数、分数和浮点型数据。数字型的定义是 NUMBER (precision, scale)，precision 表示总共的数字位数，scale 是精度，表示小数点后多少位。
- DATE 存放日期和时间。
- BLOB 存储非结构化的二进制数据，最大可存储 128TB。
- CLOB 存储字符数据，最大可存储 128TB。
- NCLOB 存储 Unicode 国家字符数据，最大可存储 128TB。
- BFILE 文件指针，指向操作系统文件，实际的文件存储在操作系统中，字段中存储的是文件定位指针。BFILE 是只读的，不参与事务控制。数据库管理员必须确保文件的存在。
- ROWID 存放表中数据行的地址（ROWID）。
- UROWID 可存储物理行 ID（Physical ROWID）和逻辑行 ID（Logical ROWID）。

有关“用户定义数据类型”，**请参考本书 16.8 节的内容**。



16.4 数据库的核心——表结构管理

在第 1 章中，我们已经定义了表，表其实是数据的“容器”。接下来，我们将深入对表进行讲解。



16.4.1 Oracle 的表类型

Oracle 有几种类型的表：

- 普通表（Ordinary Table），又叫堆组织表（Heap Organized Table）。
- 聚簇表（Clustered Table）。
- 分区表（Partitioned Table）。
- 外部表（External Table）。
- 临时表（Temporary Table）。
- 索引组织表（Index-Organized Table，IOT）。

1．普通表（Ordinary Table）

普通表就是我们平常所指的表，其数据以无序方式进行存储。

2．聚簇表（Clustered Table）

聚簇是表的一种特殊结构。一个聚簇由多个表组成，几个表共享相同的数据块。一个聚簇有一个或者多个公共的列，多个表共享这些列，我们把这样的列叫**聚簇关键字**（Cluster Key）。Oracle 把多个表的数据物理地存储在一起，以加速表的连接（Join），这是聚簇的优点。

只有创建聚簇后，才能在聚簇中创建表，这些表叫**聚簇表**。在往聚簇表中插入数据之前，必须在聚簇上创建聚簇索引。

在应用中，只有满足下面两个条件，才适合使用聚簇：

- 建立聚簇表的目的是为了查询，不是为了修改，频繁修改的表不适合创建成聚簇表；
- 查询时，常常对聚簇中的多个表进行连接。

3．分区表（Partitioned Table）

有关分区表，将在本章最后进行讲解。

4．外部表（External Table）

外部表是这样的表，数据库只存放外部表的定义，外部表的数据存放在数据库之外。创建外部表的 SQL 语句分成两部分：一部分描述列的定义；另一部分描述外部数据源如何映射（Map）成数据库的列。

外部表的数据以文件的信息存放在操作中，简单地说，外部表的数据存放在操作系统

的文件中。要想使用外部表，用户必须有数据源所在目录读的权限（这里的目录是数据库对象）。

创建外部表时，要指定访问驱动（Access Driver），访问驱动用于把外部数据读入 Oracle 中。Oracle 提供两种驱动：ORACLE_LOADER 与 ORACLE_DATAPUMP。默认的驱动是 ORACLE_LOADER。

外部表是只读的，因此不能在外部表上执行任何 DML 语句。

5．临时表（Temporary Table）

与“永久表”相对，在 Oracle 中，存在另外一种类型的表叫“临时表”。“临时表”就是指表中的数据是临时存在的。临时表的数据只存在一次会话或者一个事务中，而临时表的定义永久存在数据字典中。

我们可以使用 SQL 语句 CREATE GLOBAL TEMPORARY TABLE 创建临时表。临时表分成两类：

- 会话型的临时表（Session-Specific），会话型的临时表的数据只存在会话期间，如果用户退出登录，Oracle 会自动删除临时表中的数据。对于当前会话来说，临时表是私有的，每个会话只能修改自己的临时表数据。
- 事务型的临时表（Transaction-Specific），事务型的临时表的数据只存在事务期间，如果事务结束，Oracle 将自动删除临时表中的数据。

可以在临时表上执行 DML 语句，临时表上的 DML 语句不会产生重做日志（Redo Logs），但是会产生回滚日志（Undo Logs）。用户可以在临时表上创建索引及视图。

临时表的定义并不会因为事务的结束、会话的终止或者数据库的重起而消失，临时表的定义永久保存在数据字典中。临时表的数据存放在临时表空间中。

6．索引组织表（Index-Organized Table, IOT）

索引组织表是一类特殊的表，它将索引和表的数据存储在一起。普通表的数据以无序（Heap）的方式存放在数据库中，而索引组织表按照主键进行排序，以二叉树的形式对表的数据进行存储。二叉树中的每个索引条目用<主键值,非主键值>来表示。索引组织表不存储 ROWID，它通过主键来访问数据。索引组织表适合通过主键对数据进行访问的应用。

索引组织表有下面的优点：

- 快速的随机访问。我们都知道，索引的扫描效率非常高，索引组织表正是利用索引的这个优点。索引和表的数据存储在一起，如果对表进行更新，Oracle 只更新索引结构。
- 快速的范围扫描。索引组织表已经按照主键对数据进行排序，因此，范围扫描的速度是很快的。
- 更少的存储需求。索引数据和表的数据存储在一起，可以减少存储需求。

我们都知道，普通的索引条目只包含索引值和指向数据行的 ROWID，而索引组织表中的索引条目不仅包含索引值还包含索引数据，有时候，表的数据很大，以至于一个索引条目不能够容纳，因此，Oracle 将一条记录分成两部分进行存储：

- 索引部分。索引部分用于存放主键值、频繁访问的部分非主键值、指向溢出区的ROWID。
- 溢出区。溢出区用于存放非主键值。溢出区存放在一个溢出表空间中。用户可以指定溢出表空间。

7．哑表 DUAL 的使用

哑表是创建数据库时由 Oracle 自动创建的一张表，这个表属于 SYS 用户，该表由一列 DUMMY 构成，且该表中只存储一条记录。该表非常有用，我们将在后续的章节详细介绍。哑表的内容如下所示：

```
SQL> SELECT * FROM dual;
DUMMY
-----
X
```



16.4.2 建表——创建存储数据的“容器”

本节将介绍如何创建不同的表。

1．创建普通表（Ordinary Table）

创建普通表的例子如下所示：

```
CREATE   TABLE tb_ordernary5
(
  id     NUMBER NOT NULL PRIMARY KEY,
  name    VARCHAR2(20) DEFAULT 'OS',
  address VARCHAR2(40)
)
TABLESPACE myspace
  STORAGE
  (
    INITIAL 64K
    NEXT 1M
    MINEXTENTS 1
    MAXEXTENTS UNLIMITED
  );
```

- CREATE TABLE 创建表的关键字，后面是表的名字。
- NOT NULL 指定列不允许为空。
- PRIMARY KEY 指定该列是主键。
- DEFAULT 指定列的默认值。
- TABLESPACE 指定表存放的表空间。
- STORAGE 指定表的存储参数。

- INITIAL 指定初始区的大小。
- NEXT 指定下一个区的大小。
- MINEXTENTS 指定最少要分配多少个区。
- MAXEXTENTS 指定区的最大数量。
- 本例中，VARCHAR2(20)指定了列 name 的数据类型及长度。

2．创建聚簇表（Clustered Table）

创建聚簇表分成两个步骤。在创建聚簇表以前，必须创建聚簇（或聚簇已经存在），然后把表加入聚簇中（即创建聚簇表）。

STEP 01 创建聚簇。

```
CREATE CLUSTER tb_cluster(postcode int)
TABLESPACE myspace;
```

创建聚簇 tb_cluster，CREATE CLUSTER 表示创建聚簇，postcode 是聚簇关键字，myspace 是存放聚簇的表空间。

STEP 02 把表 student 加入聚簇 tb_cluster 中（创建聚簇表）。

```
CREATE TABLE student (
ID INT PRIMARY KEY,
NAME VARCHAR2(20) NOT NULL,
postcode INT )
CLUSTER tb_cluster(postcode);
```

STEP 03 把表 address_infor 加入聚簇 tb_cluster 中（创建聚簇表）。

```
CREATE TABLE address_infor(
postcode INT PRIMARY KEY,
name VARCHAR2(40),
detail VARCHAR2(40))
CLUSTER tb_cluster (postcode);
```

3．创建索引组织表（Index-Organized Table）

创建索引组织表的例子如下所示：

```
CREATE TABLE students(
name    VARCHAR(20) PRIMARY KEY,
id      NUMBER,
detail  VARCHAR2(100)
)
ORGANIZATION INDEX
TABLESPACE users
PCTTHRESHOLD 30
INCLUDING detail
OVERFLOW TABLESPACE myspace;
```

- PRIMARY KEY 指定主键，索引组织表中一定要有主键。

- ORGANIZATION INDEX 指定创建的表是索引组织表。
- PCTTHRESHOLD 指定溢出比例，如果超过溢出比例的限制，则溢出部分将被存储到溢出区中。
- INCLUDING 指定列名，表示从这个列以后的所有列将存储在溢出区中。
- OVERFLOW TABLESPACE 指定溢出表空间。

4．创建外部表（External Table）

下面我们将演示创建外部表的全过程。

STEP 01 创建目录。

在操作系统中创建目录 D:\temtb\data、D:\temtb\log、D:\temtb\bad，使 Oracle 对这些目录有读写权限。

STEP 02 在操作系统中创建文件。

在操作系统中创建文件 D:\exdata\temstu.txt，文件的内容如下所示：

```
50016,小张,上海,22
30011,小李,天津,24
40022,晶晶, 广州,25
30044,小宝, 六盘水,22
20055,小狐狸, 郴州,26
```

特别注意的是，外部表的数据不是存放在数据库中，而是存放在操作系统文件中（本例是 temstu.txt）。

STEP 03 以数据库管理员的身份登录数据库。

STEP 04 创建目录对象。

```
CREATE OR REPLACE DIRECTORY dat_dir
AS 'D:\temtb\data';

CREATE OR REPLACE DIRECTORY log_dir
AS 'D:\temtb\log';

CREATE OR REPLACE DIRECTORY bad_dir
AS 'D:\temtb\bad';
```


这里的目录是一个数据库对象，并不是真正的操作系统目录，但是，它们指向真正的操作系统目录。其中，D:\temtb\data 存放数据；D:\temtb\log 存放日志；D:\temtb\bad 存放坏文件。如果一条记录格式不对或者这条记录引起了 Oracle 错误，这条记录将被写到坏文件中。

STEP 05 授权用户 rawman 对目录的**读**权限。

```
SQL> GRANT READ ON DIRECTORY dat_dir TO rawman;
GRANT succeeded.
```

本例授予用户 rawman 对目录（dat_dir）**读**的权限。

```
SQL> GRANT READ,WRITE ON DIRECTORY log_dir TO rawman;
```

```
GRANT succeeded.
```

本例授予用户 rawman 对目录（log_dir）**读/写**的权限。

```
SQL> GRANT READ,WRITE ON DIRECTORY bad_dir TO rawman;
GRANT succeeded;
```

本例授予用户 rawman 对目录（bad_dir）读/写的权限。

为了使用外部表，Oracle 要求：

- 对存放数据的目录要有读的权限。
- 对存放日志文件的目录要有写的权限。
- 对存放坏文件的目录要有写的权限。



以用户 rawman 的身份登录数据库。

```
CONNECT rawman;
```



创建外部表。

```
CREATE TABLE fitness_member
(id integer ,
name VARCHAR2(14),
city VARCHAR2(30),
age int)
ORGANIZATION EXTERNAL
(
TYPE ORACLE_LOADER
DEFAULT DIRECTORY dat_dir
ACCESS PARAMETERS
(
records delimited by newline
badfile bad_dir:'empxt%a_%p.bad'
logfile log_dir:'empxt%a_%p.log'
fields terminated by ','
missing field VALUES are NULL
(id,name,city,age)

)
LOCATION ('temstu.txt')
)
PARALLEL
REJECT LIMIT UNLIMITED;
```

- fitness_member 是外部表的名字。
- ORGANIZATION EXTERNAL 表示创建的表是外部表。
- TYPE ORACLE_LOADER 指定访问的驱动是 ORACLE_LOADER。
- DEFAULT DIRECTORY dat_dir 指定数据的默认存放路径。
- records delimited by newline 一行为一条记录。

- badfile bad_dir:'empxt%a_%p.bad'指定坏文件的名字及路径。
- logfile log_dir:'empxt%a_%p.log'指定日志文件的名字及路径。
- fields terminated by '，'指定字段之间是以'，'进行分隔的。
- LOCATION (' temstu.txt)指定存放数据的文件的名字。
- REJECT LIMIT UNLIMITED 表示允许无限制的行发生错误。

STEP 08 查询外部表。

```
SELECT * FROM fitness_member;
```

SQL 语句的执行结果如下所示：

```
ID      NAME        CITY        AGE
50016   小张        上海        22
30011   小李        天津        24
40022   晶晶        广州        25
30044   小宝        六盘水      22
20055   小狐狸      郴州        26
```

对比 STEP 02 和 STEP 08，可以看出，查询的结果和文件 temstu.txt 中的内容一致。查询的数据来源于数据库外部的操作系统文件。

5．创建临时表（Temporary Table）

临时表用于临时存放中间数据，在应用程序开发中，它只担当中介的作用。有两种类型的临时表：会话型临时表（Session-Specific）和事务型的临时表（Transaction-Specific）。下面我们将演示如何创建临时表。

- 创建事务型临时表

```
CREATE GLOBAL TEMPORARY TABLE gtt2
(name varchar2(20),
id number,
bithday DATE)
ON COMMIT DELETE ROWS;
```

- 创建会话型临时表

```
CREATE GLOBAL TEMPORARY TABLE gtt4
(name varchar2(20),
id number,
bithday DATE)
ON COMMIT PRESERVE ROWS;
```

其中，CREATE GLOBAL TEMPORARY TABLE 表示创建临时表；ON COMMIT 指定临时表的类型：DELETE ROWS 是事务型临时表；PRESERVE ROWS 是会话型临时表。

6．表的克隆

如果已经存在一个表，可以创建另外一个和该表一模一样的表，我们把这个过程叫做

表的“克隆”。接下来，我们将演示表的克隆。

STEP 01 准备要被克隆的表。

准备要被克隆的表 stb，我们将利用该表克隆出其他两个表 t_clone1 与 t_clone2。

```
CREATE TABLE stb
(id number,
name varchar2(30));

INSERT INTO stb
VALUES(1111,'水晶');

COMMIT;

SELECT * FROM stb;
ID      NAME
----------------------------
1111        水晶
```

表 stb 中存有一条数据。

STEP 02 克隆表 t_clone1。

```
CREATE TABLE t_clone1
AS SELECT * FROM stb
WHERE 1=0;

SELECT * FROM t_clone1;
```

克隆出表 t_clone1，表 t_clone1 的结构和 stb 的结构相同，但是 t_clone1 中没有任何数据。也就是说，t_clone1 和 stb 仅仅结构相同。

STEP 03 克隆表 t_clone2。

```
CREATE TABLE t_clone2
AS SELECT * FROM stb;

SELECT * FROM t_clone2;
ID      NAME
----------------------------
1111    水晶
```

克隆出表 t_clone2，表 t_clone2 和 stb 完全相同（包括结构和数据）。



16.4.3 修改表——修改“容器”的属性

创建表时，需要指定表的属性；即使表已经存在，我们仍然可以修改表的属性。修改表的属性其实就是修改表的定义。

1．添加列

往表 tb_ordernary5 中添加列 lenth。

```
ALTER TABLE tb_ordernary5
ADD (lenth NUMBER (8,3));
```

2．列的重命名

把表 tb_ordernary5 的列 lenth 重命名成 new_lenth。

```
ALTER TABLE tb_ordernary5
RENAME COLUMN lenth TO new_lenth;
```

3．改变列的属性

使用 ALTER TABLE ... MODIFY 命令改变列的属性（包括列的数据类型、数据精度、默认值等）。

- 修改列的宽度

```
ALTER TABLE tb_ordernary5
MODIFY (new_lenth NUMBER(4,1));
```

- 修改列的默认值

```
ALTER TABLE tb_ordernary5
MODIFY (new_lenth DEFAULT 5);
```

- 设置列不能为空

```
ALTER TABLE tb_ordernary5
MODIFY (new_lenth NOT NULL);
```

- 对列进行加密

```
ALTER TABLE tb_ordernary5
MODIFY (new_lenth ENCRYPT USING '3DES168');
```

使用加密算法 3DES168 对列 new_lenth 进行加密。

- 对列进行解密

```
ALTER TABLE tb_ordernary5
MODIFY (new_lenth DECRYPT);
```


4．为表手工分配一个新的区（Extent）

为表 tb_ordernary5 新分配一个大小是 50KB 的区（EXTENT）。

```
ALTER TABLE tb_ordernary5
ALLOCATE EXTENT (SIZE 50K);
```

5．删除列

删除表 tb_ordernary5 的列 address。

```
ALTER TABLE tb_ordernary5 DROP (address);
```

也可以一次删除多列，如：

```
ALTER TABLE tb_ordernary5 DROP (name,address);
```



16.4.4 表管理小技巧

1．把表移动到一个新的段（Segment）

把表 tb_ordernary5 移动到一个新的段。

```
ALTER TABLE tb_ordernary5 MOVE
     STORAGE ( INITIAL 20K
               NEXT 40K
               MINEXTENTS 2
               MAXEXTENTS 20
               PCTINCREASE 0 );
```

在移动表的时候，还可以修改表的存储属性。

2．把表移动到其他表空间

把表 tb_ordernary5 移动到表空间 myspace2。

```
ALTER TABLE tb_ordernary5 MOVE TABLESPACE  myspace2;
```

3．锁住你的表

➢ 示例 1：往表上加行级锁。

```
LOCK TABLE studentbook
IN EXCLUSIVE MODE
NOWAIT;
```

上述代码在表 studentbook 上加上排他锁。

➢ 示例 2：锁住表中指定的行。

假设表 tab_locktest 中有下面的数据行：

```
NAME        ID      ADRRESS
许小明      111     北京
刘小刚      222     贵州
汪伦        333     台湾
许晓晶      444     湖南
SELECT * FROM tab_locktest WHERE NAME = '许%' FOR UPDATE;
```

执行上面的 SQL 语句将锁住名字是许小明、许晓晶的两条记录。

➢ 示例 3：解锁。

```
COMMIT;
```

上述命令提交事务，锁将被解开。

➢ 示例 4：解锁。

```
ROLLBACK;
```

上述命令回滚事务，锁也将被解开。



知识点索引——锁（Lock）

Oracle 把锁分成行级锁（Row Locks，TX）和表级锁（Table Locks，TM）。

行级锁（TX）

行级锁(TX)能够阻止两个事务对相同数据行的改变。当一个事物修改一行数据，将会获得一个行级锁。执行 INSERT、UPDATE、DELETE、SELECT…FOR UPDATE 将会自动获得行级排他锁（Exclusive Row Lock），一旦一个事务获得该锁，其他事务将无法修改相同的数据行，直到该事务被提交或者回滚。

表级锁（TM）

表级锁用于锁住整个表，而不是单个表中特定的行。表级锁用于防止 DML 和 DDL 的冲突。Oracle 提供下面的表级锁。

- ✓ 行共享（ROW SHARE），允许其他用户并发访问被锁住的表，但是禁止以行排他访问锁住整个表。
- ✓ 行排他（ROW EXCLUSIVE），行排他和行共享相同，只是禁止在该表上加共享锁（SHARE）。行排他锁通过更新（UPDATING）、插入（INSERTING）、删除（DELETING）自动获得。
- ✓ 共享更新（SHARE UPDATE），共享更新和行共享（ROW SHARE）相同。
- ✓ 共享（SHARE）允许并发查询，但是禁止对锁住的表进行更新。
- ✓ 共享行拍他（SHARE ROW EXCLUSIVE），允许查看数据，但是禁止其他用户以共享（SHARE）模式锁住表。
- ✓ 排他（EXCLUSIVE），允许对锁住的表进行更新，但是禁止其他任何操作。

使用 LOCK TABLE 命令锁住表或者一个表的某个分区。LOCK TABLE 对表的读没有任何影响。如要锁住其他用户的表，你必须拥有 LOCK ANY TABLE 权限。

锁的管理需要很高的技巧，建议不要手动对表加锁！

4．让一个列自动增长

Oracle 没有提供一种便捷的方法实现表中某列的自动增长，但是，我们可以使用一种变通的方法以实现表中某列的自动增长。下面，我们将通过一个例子演示如何使一个列的值自动增长。



创建订单表 myorder。

```
CREATE TABLE myorder
```

```
(order_NO NUMBER,
order_name VARCHAR2(20),
order_desc VARCHAR2(20));
```

本例将使表中的订单编号 order_NO 自动增加。其中，order_NO 表示订单编号；order_name 表示订单的名字；order_desc 表示订单描述。



创建序列 seq_order_NO。

```
CREATE SEQUENCE seq_order_NO
START WITH 1
MAXVALUE 9999999999999999999999999999
MINVALUE 1
NOCYCLE
CACHE 100;
```

其中，START WITH 指定序列的起始值；MAXVALUE 指定序列的最大值；MINVALUE 指定最小值；NOCYCLE 表示序列到达最大值以后不再循环，这样可以防止序列号重复；CACHE 100 表示缓存 100 个序列值，这样可以提高序列的访问速度。

有关序列的介绍，**请参考本书第 18 章的内容**。



创建触发器 trigger_myorder。

```
CREATE OR REPLACE TRIGGER trigger_myorder
  BEFORE INSERT ON myorder
  FOR EACH ROW
BEGIN
  SELECT seq_order_NO.NEXTVAL INTO :NEW.order_NO FROM dual;
END;
```

其中，BEFORE INSERT 表示插入前触发；FOR EACH ROW 表示对每一行都触发，在此，使表 myorder 的字段 order_NO 自动增加；NEXTVAL 表示取序列（seq_order_NO）的下一个值。

有关触发器的概念，**请参考本书 19.3 节的内容**。



往表 myorder 中插入数据。

```
INSERT INTO myorder
  (order_name, order_desc)
VALUES
  ('光盘采购', '采购公司需要的关盘');
INSERT INTO myorder
  (order_name, order_desc)
VALUES
  ('大米采购', '要求来自泰国的大米');
COMMIT;
```

插入行数据时，并没有指定列 order_NO 的值，列 order_NO 的值应该隐式增加。



查看字段 order_NO 是不是自动增加。

```
SQL> SELECT * FROM myorder;
```

```
  ORDER_NO ORDER_NAME          ORDER_DESC
---------- ------------------- -------------------
         1 光盘采购            采购公司需要的关盘
         2 大米采购            要求来自泰国的大米
```

从上面可以看出，order_NO 从 1 到 2 自动增加。

5．把一个表放入只读模式

把表 tb_ordernary5 放入只读模式。

```
ALTER TABLE tb_ordernary5  READ ONLY;
```

可以利用下面的命令使表回到可读写模式：

```
ALTER TABLE tb_ordernary5 READ WRITE;
```

不允许在只读表上执行的操作有：

- 所有 DML 操作
- 截断表（TRUNCATE TABLE）
- 锁住指定的数据（SELECT FOR UPDATE）
- 修改表的属性（ALTER TABLE ADD/MODIFY）
- 把列置于不可用状态（ALTER TABLE SET COLUMN UNUSED）
- 分区操作（ALTER TABLE DROP/TRUNCATE/EXCHANGE (SUB)PARTITION）
- ALTER TABLE UPGRADE INCLUDING DATA
- ALTER TYPE CASCADE INCLUDING

在线重定义（Online Redefinition）

- 闪回表（FLASHBACK TABLE）



知识点索引——只读表

如果一个表（如配置信息表）在应用程序部署以后，不需要改变，可以把该表放入只读模式。被置于只读模式的表不允许被修改。

6．判断表的存在性

使用 SQL 语句 SELECT * FROM　user_name.table_name 判断表的存在性，如果表不存在，会提示错误“ORA-00942: table or view does not exis”。表不存在有几个原因：

- 你无权访问该表
- 表名的大小写引起的错误
- 该表确实不存在

7．判断用户下所有表是否含有日期字段

找出当前用户模式下，哪些表的哪些列的数据类型是日期型，并列出表及相关的列。

```
SQL> SELECT c.table_name, c.column_name
  2    FROM user_tab_cols c
```

```
  3  WHERE c.data_type = 'DATE'
  4  ORDER BY table_name
```

SQL 语句的执行结果如下所示：

```
TABLE_NAME                     COLUMN_NAME
------------------------------ ------------------------------
TB_APPLY                       APPLY_DATE
TB_DATE_TEST                   BIRTH
TB_MAINTENCE                   START_DATE
TB_MAINTENCE                   END_DATE
```

本例中，共有 4 个表含有日期型的列。其中，表 TB_MAINTENCE 有两列（START_DATE、END_DATE）都为日期型。

8．查询出组成表的列

有两种方法均可以知道组成表的列及列的数据类型等信息。

➢ 用 DESC 命令

在 SQL*Plus 中，执行命令 DESC，可以知道表的结构。

```
SQL> DESC tb_date_test
Name     Type   Nullable Default Comments
-----    ------ -------- ------- --------
ID       NUMBER      Y
BIRTH    DATE        Y
```

本例中，表 tb_date_test 由两列组成，列 ID 的数据类型是 NUMBER，BIRTH 的数据类型是 DATE。

➢ 查询视图 ALL_TAB_COLUMNS（或 DBA_TAB_COLUMNS）

```
SQL> SELECT COLUMN_NAME,DATA_TYPE,NULLABLE,DATA_DEFAULT
  2  FROM all_tab_columns
  3  WHERE table_name='TB_DATE_TEST'
```

SQL 语句的执行结果如下所示：

```
COLUMN_NAME           DATA_TYPE    NULLABLE DATA_DEFAULT
------------------------------ ------------ -------- ------------
BIRTH                          DATE        Y
ID                             NUMBER      Y
```

其中，COLUMN_NAME 表示列的名字；DATA_TYPE 表示列的数据类型；NULLABLE 表示该列是否允许为空；DATA_DEFAULT 表示默认值。

9．计算一个表的大小

通过视图 DBA_EXTENTS 的 bytes 字段可以计算一个表的大小。计算表 S_EMPLOYEE 的大小的命令如下所示：

```
SQL> SELECT segment_name, SUM(bytes) / 1024 / 1024 || 'M'
  2   FROM dba_extents
  3  WHERE segment_name = 'S_EMPLOYEE'
  4  GROUP BY segment_name;
```

SQL 语句的执行结果如下所示：

```
SEGMENT_NAME  SUM(BYTES)/1024/1024||'M'
----------------------------------------------------------------
S_EMPLOYEE       2M
```

表 S_EMPLOYEE 的大小是 2MB。

可以使用类似于下面的命令给表添加新的区。

```
ALTER TABLE S_EMPLOYEE
ALLOCATE EXTENT (SIZE 1M);
```



16.4.5 删除表——让整个表从数据库中彻底消失

DROP 的含义是删除表的结构和数据，并回收该表占用的空间，需要注意的是，DELETE 仅仅删除表中的数据，对表的结构没有任何影响。DROP 是 DDL 语句，DELETE 是 DML 语句。

- 分步骤彻底删除表



删除普通表（并未真正删除表，只是把该表放在回收站中）。

```
DROP TABLE student_infor;
```

删除带约束的表。

```
DROP TABLE tb_date_test CASCADE CONSTRAINTS;
```



查看回收站。

```
SQL> SHOW RECYCLEBIN
ORIGINAL NAME    RECYCLEBIN NAME            OBJECT TYPE   DROP TIME
---------------- ------------------------------ ------------
STUDENT_INFOR   BIN$6IW9QnJsT7G+llYVjdfdvA==$0  TABLE   2009-09-01:15:39:40
TB_DATE_TEST    BIN$D1xFqlQcQamgV4tgp7yoZA==$0  TABLE   2009-09-04:19:26:40
```


从回收站中，可以看到已经被删除的两个表，还可以看到表原来的名字、在回收站中的名字及删除日期。

也可以使用 SQL 语句 SELECT * FROM RECYCLEBIN 查看回收站中的对象。



清空回收站中指定的表（会释放表占用的空间）。

```
PURGE TABLE "BIN$6IW9QnJsT7G+llYVjdfdvA==$0";
```

想清空回收站中所有的表，请执行下面的命令。

```
PURGE RECYCLEBIN;
```

➢ 一次性彻底删除表

STEP 01 删除表。

```
DROP TABLE tb_maintence PURGE;
```

PURGE 选项指示一次性彻底删除表，不把该表扔到回收站中。

STEP 02 查询回收站。

```
SQL> SHOW RECYCLEBIN
SQL>
```

回收站中没有表 tb_maintence 的任何痕迹，说明表已经被彻底删除。

知识点索引——回收站（Recycle Bin）

在 Oracle 中，也有回收站的概念，这里的回收站在功能和使用上和 Windows 的回收站类似。当删除一个表时，Oracle 并没有真正删除该表，而是把该表重命名，然后扔到回收站中。使用 PURGE 命令来清空回收站，将会释放表占用的空间，把这些空间返回给表空间，成为表空间中的自由空间。

一旦空间吃紧，Oracle 会自动清空回收站中的对象。在 Oracle 清空前，我们可以找回被删除的对象，就是把被删除的对象从回收站中捡回来。

从回收站中捡回被删除的表 instructor：

```
FLASHBACK TABLE instructor TO BEFORE DROP
```

从回收站中捡回被删除的表 instructor，并重命名该表：

```
FLASHBACK TABLE instructor TO BEFORE DROP
RENAME TO instructor_restore;
```

可以使用 DESC 命令查看表 instructor_restore 是否已经捡回。

16.5 数据库的核心——表数据管理

本节主要讲述表中数据的管理，并展示数据管理的一些技巧，许多例子来源于实际的项目开发。

16.5.1 往“容器”中装入数据（INSERT 语句演示）

本节主要介绍数据插入，并演示插入技巧。

1. INSERT 语句的基本功能

INSERT 语句用于把数据添加到表、视图的基表、表的一个分区或者对象表中。要想

往其他用户拥有的表中插入数据，执行插入的用户必须拥有表上的 INSERT 权限（对象权限）。如果一个用户拥有 INSERT ANY TABLE 权限（系统权限），则该用户可以往任何用户的表中插入数据。

INSERT 语句的基本语法如下所示：

```
INSERT INTO 表名（列1，列2，列3，...）
VALUES(值1，值2，值3，...);
```

其中，INSERT INTO 表示往指定的表中插入数据，后面是表名；“列 1，列 2，列 3，...”指定要被插入数据的列，也可不指定列名；VALUES 指定插入值。

示例 1：往表中插入数据（指定列名）。

```
INSERT INTO departments(DEPARTMENT_ID,DEPARTMENT_NAME,MANAGER_ID,LOCATION_ID)
VALUES (220, 'Recreation', 111, 7200);
```

指定的列和 VALUES 中指定的值一一对应，如 DEPARTMENT_ID 对应 220。

未指定列名的列将被插入默认值或者空值（NULL）。

示例 2：往表中插入数据（未指定列名）。

```
INSERT INTO departments
VALUES (220, 'Recreation', 111, 7200);
```

指定的值和表中列的顺序必须一一对应。本例中，虽然没有指定列名 MANAGER_ID，但是，值 111 必须和表的列 MANAGER_ID 对应。

2．插入字符&

■ 试验准备

创建表 tb_char，下面我们将往该表中插入含有&符号的字符串。

```
CREATE TABLE tb_char
(name varchar2(20));
```

■ 实施试验

➢ 方法 1：设置 SQL*Plus 环境变量。

STEP 01 在 SQL*Plus 中，执行下面的 SQL 语句。

```
set define off
```

STEP 02 插入含有&符号的字符串。

```
INSERT INTO tb_char
VALUES('CHINA&AMERICA');
```

➢ 方法 2：使用函数 chr。

```
INSERT INTO tb_char VALUES (chr(38));
```

■ 查询插入值

```
SQL> SELECT * FROM tb_char;
```

```
NAME
--------------------
&
CHINA&AMERICA
```

从查询结果可以看出，上述两种方法均可插入特殊字符&。

3．根据一个表往另外一个表中插入数据

本节将演示如何根据一个表往另外一个表中插入数据。



创建数据源表 t_source。

```
CREATE TABLE t_source
(id number ,name varchar2(20),address varchar2(20));
```



往数据源表 t_source 中插入数据。

```
INSERT INTO t_source
VALUES(111,'小明','上海');

INSERT INTO t_source
VALUES(121,'小张','北京');

INSERT INTO t_source
VALUES(121,'小李','上海');
COMMIT;
```



查询数据源表。

```
SQL> SELECT * FROM t_source;
       ID NAME                 ADDRESS
---------- -------------------- --------------------
      111 小明                 上海
      121 小张                 北京
      121 小李                 上海
```

从查询结果可知，表 t_source 中共有 3 条记录。



创建目标表。

```
CREATE TABLE t_target
(id number ,name varchar2(20),address varchar2(20),phone varchar2(30));
```



从数据源表 t_source 往目标表 t_target 中插入数据。

```
INSERT INTO t_target(id,name)
SELECT id, name
FROM t_source t
WHERE t.address='上海';
COMMIT;
```



查看目标表中的数据。

```
SQL> SELECT * FROM t_target;
```

```
        ID NAME                ADDRESS            PHONE
---------- ------------------- --------------------
       111                     小明
       121                     小李
```

从目标表中可知，只有两条记录被插入目标表中。

4．根据一张表，有条件地往另外一张表或视图中插入数据或者更新数据（MERGE）

当同步两张表中的数据时，我们可以有条件地使用一张表（**源表**）中的数据去更新（UPDATE）另外一张表中的数据（**目标表**）；如果要更新（UPDATE）的数据不存在于**目标表**中，我们可以有条件往**目标表**中插入（INSERT）**目标表**中不存在的数据行。要想一条 SQL 语句同时完成更新（UPDATE）和插入（INSERT）的过程，我们只需使用 Oracle 的 MERGE 语句即可。

接下来，我们将以一个示例演示 MERGE 的使用。本例中的目标表是 t_bonus，用于存放员工的奖金；源表是 s_employee，存放员工的基本工资。本例将根据员工的基本工资（salary，在源表中）去设置员工的奖金（bonus，在目标表中）。详细过程说明如下（请参考本书配套光盘\script\16-1.SQL）。

STEP 01 准备目标表 t_bonus。

ⓐ 创建目标表。

```
CREATE TABLE t_bonus
(employee_id VARCHAR2(10),
bonus NUMBER);
```

该表存放员工奖金。其中，employee_id 表示员工的 ID；bonus 表示员工现在的奖金金额。

ⓑ 往目标表中插入数据。

```
INSERT INTO t_bonus
VALUES(111,221);

INSERT INTO t_bonus
VALUES(211,1221);

INSERT INTO t_bonus
VALUES(212,1300);
```

STEP 02 准备数据源（源表）。

ⓐ 创建源表。

```
CREATE TABLE s_employee
(employee_id VARCHAR2(10),
salary NUMBER);
```

其中，employee_id 表示员工的 ID；salary 表示员工的基本工资。

ⓑ 插入数据。

```
INSERT INTO s_employee
VALUES(111,221);

INSERT INTO s_employee
VALUES(211,1221);

INSERT INTO s_employee
VALUES(212,1300);

INSERT INTO s_employee
VALUES(611,2222);

INSERT INTO s_employee
VALUES(613,3300);
```


执行 MERGE 命令前，查看目标表的数据。

```
SELECT * FROM t_bonus;
```

SQL 语句的执行结果如下所示：

```
EMPLOYEE_ID     BONUS
----------- ------------------------------
111              221
211             1221
212             1300
3 rows selected
```


执行 MERGE 命令前，查看源表的数据。

```
SELECT * FROM s_employee;
```

SQL 语句的执行结果如下所示：

```
EMPLOYEE_ID    SALARY
----------- ---------------------------------
111              221
211             1221
212             1300
611             2222
613             3300
5 rows selected
```


执行 MERGE 命令。

```
MERGE INTO t_bonus T
USING (SELECT employee_id, salary FROM s_employee) S
– USING 指定数据源，数据源可以来自一个查询，可以来自表，也可以来自视图。
ON (T.employee_id = S.employee_id)
–指定一个条件，将根据此条件决定是执行更新还是插入操作
WHEN MATCHED THEN UPDATE SET T.bonus = S.salary + 40
```

```
    -顾名思义，只有满足 ON 指定的条件的记录才被更新。
    WHERE T.bonus>1000
    --WHERE 条件可以针对目标表，也可以针对源表。满足此条件才会执行更新操作。
    DELETE WHERE (T.bonus > 1300)
    --DELETE WHERE 针对目标表，用于删除目标表中满足条件的记录。特别要注意的是，只删除更新以后的值。只有同时满足 DELETE WHERE 和 ON 子句的记录才被删除。
    WHEN NOT MATCHED THEN INSERT (T.employee_id, T.bonus)
    --不满足 ON 指定的条件的记录才被插入。employee_id，bonus 表示指定插入列。
    VALUES (S.employee_id, S.salary+20)
    --指定插入值
    WHERE (S.salary > 3000)
    --这里的 WHERE 条件只针对数据源。满足此条件的记录才允许被插入。
```

STEP 06 查看执行 MERGE 命令后的目标表。

```
SELECT * FROM t_bonus;
```

SQL 语句的执行结果如下所示：

```
EMPLOYEE_ID     BONUS
----------- ----------
111              221
211             1261
613             3320
3 rows selected
```

- ID 是 111 的员工，既不满足插入条件，也不满足更新条件，因此，该员工的奖金没有任何改变，还是 221 元。
- ID 是 211、212 的两位员工当前奖金都大于 1000 元（T.bonus>1000），因此员工 211 的奖金是：基本工资(1221 元)+40 元=1261 元，员工 212 的奖金是：基本工资(1300 元)+40 元=1340 元。因为员工 212 更新后的工资（1340 元）大于 1300 元(T.bonus > 1300)，因此该记录被从目标表中删除。
- ID 是 613 的员工以前没有奖金(也就是该员工的信息不存在于目标表 t_bonus 中)，并且该员工工资是（3300）大于 3000（S.salary > 3000)，因此该员工可以调奖金，该员工的奖金是：基本工资（3300）+20，也就是 3320。

5．插入日期

日期的插入本来是一件很简单的事，但是 Oracle 把它变得稍微有些复杂。插入日期的例子说明如下。

STEP 01 创建测试表。

```
CREATE TABLE tb_date
(cdl date);
```

STEP 02 插入当前的系统日期。

```
INSERT INTO tb_date
```

```
VALUES(SYSDATE);
```

其中，SYSDATE 表示当前的系统日期。

STEP 03 插入另一指定日期。

```
INSERT INTO tb_date
VALUES(to_date('2009-09-9 12:27:10','YYYY-MM-DD HH24:MI:SS'));
```

插入日期，需要用到 TO_DATE 函数，该函数把字符型数据转换成日期型。

在 Oracle 中，日期包括时间（就是小时、分钟、秒）。YYYY-MM-DD HH24:MI:SS 用于指定日期的格式。YYYY 表示年，MM 表示月，DD 表示号数，HH24 表示 24 小时制，MI 表示分，SS 表示秒。

6．插入 NULL

空值（NULL）表示未知，并不是表示空格，NULL 和空格是两个概念。插入空值的例子说明如下。

STEP 01 创建测试表。

```
CREATE TABLE t_null
(name varchar2(10),
age number);
```

STEP 02 插入数据。

```
INSERT INTO t_null
VALUES('王晶',null);
COMMIT;
```

因为“王晶”的年龄未知，因此年龄被置成空值（NULL），表示未知。

STEP 03 查询插入数据。

```
SQL> SELECT * FROM t_null ;
 NAME            AGE
---------- ------------------------------
王晶
```

16.5.2 修改“容器”中的数据（UPDATE 语句演示）

UPDATE 语句用于对表（包括视图、物化视图）中已经存在的值进行修改。若想修改表中数据，用户必须有表上的 UPDATE 权限或者拥有系统权限 UPDATE ANY TABLE。当然，也可以更新远程数据库中的表。UPDATE 的语法如下所示：

```
UPDATE table_owner.table_name
SET column1 =values1, column2 =values2, column3 =values3…
WHERE;
```

- table_owner 是拥有要被更新的表的用户。

- table_name 是要被更新的表。
- column1、column2、column3 是要被更新的列，其他列的值保持不变。
- values1、values2、values3 将用来更新指定的列。values1 和 column1、values2 和 column2、values3 和 column3 是一一对应的关系。
- WHERE 指定限制条件，只有 WHERE 指定的行才被更新，其他的行保持不变。

示例 1：更新单列

```
UPDATE employees
  SET location = '上海'
  WHERE id = '111';
```

把编号是 111 的员工的工作地点改成上海。

示例 2：更新多列

```
UPDATE employees SET
   salary = salary + 1000,  location=null
   WHERE age>50;
```

把年龄大于 50 岁的员工的工资加 1000 元，工作地点改成未知（NULL）。

1．利用子查询对表进行更新

在应用程序开发中，根据一张表的内容去更新另外一张表是常有的事。接下来，我们通过一个例子讲述如何用一个表的内容去更新另外一张表。

📖 数据准备阶段

有一张学生信息表叫 source，ID 表示学生的学号，MAJOR 是学生的专业，该表的数据如下所示：

```
ID      MAJOR
10003   Economics
10006   Economics
10008   source
10005   source
1111    source
10002   Computer Science
10010   Nutrition
8888    8888
```

另外一张表叫 target，ID 表示学生的学号，MAJOR 是学生的专业，表中数据如下所示：

```
ID      MAJOR
10003   Economics
10006   Economics
10008   History
10005   Music
1111    History
```

```
10002   Computer Science
10010   Nutrition
9999    9999
```

📖 **任务目标**

要根据学号（ID），使表 target 中具有相同学号（ID）的学生的专业（MAJOR）和表 source 中的相同，即使表 target 中的内容变成如下形式。

```
ID      MAJOR
10003   Economics
10006   Economics
10008   source
10005   source
1111    source
10002   Computer Science
10010   Nutrition
9999    9999
```

📖 **任务实现**

要完成本例的任务，可以有三种方法，三种方法异曲同工。

■ 方法 1

```
UPDATE target
SET major=(SELECT source.major FROM source WHERE source.id=target.id)
WHERE target.id=(SELECT source.id FROM source  WHERE source.id=target.id)
```

■ 方法 2

```
UPDATE target
SET major=(SELECT source.major FROM source WHERE source.id=target.id)
WHERE target.id in(SELECT source.id FROM source)
```

■ 方法 3

```
UPDATE target
SET major=(SELECT source.major FROM source WHERE source.id=target.id)
WHERE exists ( SELECT source.id FROM source WHERE source.id=target.id )
```




16.5.3 查询“容器”中的数据（SELECT 语句演示）

SELECT 语句用于从一个或者多个表中检索数据。当然，也可以从对象表（Object Tables）、视图（Views）、对象视图（Object Views）或者物化视图（Materialized Views）中检索数据。对开发人员来说，经常要与数据库进行交互，经常需要从数据库中检索数据。SELECT 语句可能是使用频率最高的 SQL 语句之一。有时，我们需要利用 SELECT 语句的诸多技巧实现复杂的程序功能。SELECT 语句的常用语法如下：

```
SELECT column1, column2, column3,…
```

```
FROM table_name
WHERE …
```

以上的 SQL 语句表示从表（table_name）中检索列（column1、column2、column3）的值，WHERE 用于指定限定条件，表示只检索满足条件的记录。

1．从表中检索指定列的数据（SELECT 子句）

示例 1：从表中检索指定列的数据

```
SQL> SELECT name FROM t_null;
NAME
---------- ----------
王晶
冯大纲
```

本例只显示列 name 的数据。

示例 2：从表中检索所有列的数据

```
SQL> SELECT * FROM t_null;
NAME            AGE
---------- ----------
王晶
冯大纲               40
```

本例显示表中所有列的数据。星号（*）代表所有的列。

2．列和表的别名

➢ 列的别名

在列后指定另外一个名字，我们把这个名字叫做“列的别名”。列的别名使列名的显示更加友好。使用列的别名可以简化 SQL 脚本的书写。

```
SQL> SELECT name A,age B FROM t_null;
A                B
---------- -------------------------
小布丁           19
```

从显示可以看出，列 name 的别名是 A，列 age 的别名是 B。

也可以使用类似于下面的方式指定列的别名。

```
SELECT name "A",age "B" FROM t_null;
SELECT name as "A",age as "B" FROM t_null;
```

➢ 表的别名

在表后面指定另外一个名字，我们把这个名字叫做“表的别名”，例如：

```
SELECT T.name A,T.age B
FROM t_null T
WHERE T.name='王晶';
```

表 t_null 的别名是 T，在 FROM 子句中指定表的别名。表的别名可以在当前的 SQL 语句中使用，如在本例中，别名 T 被用在 WHERE 子句中。

别名可以减少冗余字符的输入，降低 SQL 语句的复杂程度，使 SQL 语句更加易于维护和调试。

3．从多表中检索数据（FROM 子句）

有时候，查询的结果来自多个表，这就需要我们把多个表进行关联，多表关联的例子如下所示。

Employees 是员工信息表，Employees 表的数据如下所示：

```
Name        dept_id
刘晓宇      1211
张大明      2241
```

其中，Name 表示员工姓名；dept_id 表示员工所在部门的编号。

dept_infor 是部门信息表，dept_infor 表的数据如下所示：

```
dept_name       dept_id
销售部          3277
开发部          1211
拓展部          2241
```

执行多表查询。

```
SELECT A.name,B.dept_name
FROM employees A,dept_infor B
WHERE A.dept_id=B.dept_id;
```

SQL 语句的执行结果如下所示：

```
A.NAME      B.DEPT_NAME
刘晓宇      开发部
张大明      拓展部
```

本例通过关联两个表的部门编号（dept_id），查出每个员工的信息及员工所在的部门。

4．行数统计（COUNT）

COUNT 函数用于计算查询返回的行数。COUNT 函数的例子说明如下。

表 t_null 中存在如下数据：

```
SQL> SELECT name,age FROM t_null;
NAME            AGE
---------- ----------
小布丁           18
小不点           18
大猩猩           29
王晶
```

要注意的是，NAME=“王晶”这条记录中，AGE 的值为 NULL。

➢ 计算该表中所有的记录（这些记录包括重复值和 NULL 值）

```
SQL> SELECT COUNT(*) FROM t_null;
  COUNT(*)
------------------------
         4
```

该表中共有 4 条记录。星号（*）表示返回所有记录数。

➢ 计算该表的记录数（不含 NULL 值）

```
SQL> SELECT COUNT(age) FROM t_null;
COUNT(AGE)
----------------------------
         3
```

NAME=“王晶”这条记录中，AGE 值为 NULL，因此该记录被 COUNT 函数忽略掉，所以只有 3 条记录。

➢ 计算该表的记录数（含重复值）

```
SQL> SELECT COUNT(DISTINCT age) FROM t_null;
 COUNT(DISTINCTAGE)
-----------------------------------------
                2
```

该表中，有两条记录的 AGE 值重复，DISTINCT 会去掉重复值，并且有一条记录的 AGA 值为空（NULL），因此有两条记录被忽略，最终的记录数是 2。

5．过滤多余的数据行（WHERE 子句）

一个表中有许多记录，我们往往只需要其中的某些记录，在检索数据时，使用 WHERE 子句过滤掉不需要的记录，只有满足 WHERE 设定条件的记录才被检索出来。

查询名字是“冯大纲”的人的信息。

表 t_null 中的全部数据如下所示。

```
SQL> SELECT * FROM t_null;
NAME            AGE
---------- ----------
王晶
冯大纲          40
```

只检索满足条件的记录。

```
SELECT * FROM t_null
WHERE name='冯大纲';
NAME            AGE
---------- ----------
冯大纲          40
```

WHERE name='冯大纲' 做了限定，只检索出姓名是“冯大纲”的信息。

WHERE 中可以包括大于号（>）、小于号（<）、等于号（=）、加号（-）、减号（+）等运算符。

6．获取数据的前 *n* 行（TOP *n*）

无论是在 SQL Server 中还是 Oracle 中，都会有返回前 *n* 条记录的查询需求，也就是我们所说的“TOP *n*”。SQL Server 中 TOP *n* 的实现如下例所示：

```
SELECT TOP 5 * FROM kk;
```

本例返回查询结果的前 5 条记录。TOP 是 SQL Server 的关键字，后面是数字，表示返回前多少条记录。

ROWNUM 是 Oracle 的伪列（Pseudocolumn），检索数据时，Oracle 会隐含、自动地对每条取出的记录进行编号。使用 ROWNUM 可以对指定的数据行进行操作。ROWNUM 是 Oracle 中实现“TOP *n*”的必由之路。

TOP *n* 的例子说明如下。

有一查询，显示 4 条记录。

```
SQL> SELECT name,age
  2  FROM t_null;
NAME            AGE
---------- ----------
小布丁           19
小不点           18
大猩猩           29
王晶
```

取出上面结果集的前 3 条记录。

```
SQL> SELECT name,age FROM t_null
  2  WHERE ROWNUM<=3;
NAME            AGE
---------- ----------
小布丁           19
小不点           18
大猩猩           29
```

ROWNUM<=3 限定查询，只返回结果集的前 3 条记录。

需要注意的是 ROWNUM 是 Oracle 的关键字。

7．数据排序（ORDER BY 子句）

使用 ORDER BY 对检索出来的数据进行排序，ORDER BY 后面指定排序的字段，可以按照单个字段进行排序，亦可按照多个字段排序。按照多个字段进行排序时，先按照第一个字段进行排序，再按照第二个字段进行排序，以此类推。

表 t_null 中有如下数据：

```
SQL> SELECT * FROM t_null;
```

```
NAME            AGE
---------- ----------
小布丁           19
小不点           18
大猩猩           29
```

其中，AGE 表示年龄。

对表 t_null 中的数据根据年龄（age）按照升序（ASC）进行排序。

```
SQL> SELECT name, age FROM t_null
  2  ORDER BY  age ASC;
NAME            AGE
---------- ----------
小不点           18
小布丁           19
大猩猩           29
```

也可以省略关键字 ASC（如下所示），实现的效果相同。

```
SELECT name, age FROM t_null
ORDER BY  age;
```

对表 t_null 中的数据根据年龄（age）按照降序（DESC）进行排序。

```
SQL> SELECT name, age FROM t_null
  2  ORDER BY  age DESC;
NAME            AGE
---------- ----------
大猩猩           29
小布丁           19
小不点           18
```

8．查询公司员工的最高工资——函数 MAX

数据来源于本书配套光盘\script\16-2.SQL。本例主要练习函数 MAX 的使用。MAX 函数用于求最大值。例如，求员工的最高工资：

```
SQL> SELECT MAX(salary) FROM group_employees;
MAX(SALARY)
--------------------
      5000
```

该公司员工的最高工资是 5000 元。

9．查询公司员工的最低工资——函数 MIN

数据来源于本书配套光盘\script\16-2.SQL。本例主要练习函数 MIN 的使用。MIN 函数用于求最小值。例如，求员工的最低工资：

```
SQL> SELECT MIN(salary) FROM group_employees;
MIN(SALARY)
```

```
-----------------
      740
```

该公司员工的最低工资是 740 元。

10．查询公司员工的平均工资——函数 AVG

数据来源于本书配套光盘\script\16-2.SQL。本例主要练习函数 AVG 的使用。AVG 函数用于求平均值。例如，求员工的平均工资：

```
SQL> SELECT AVG(salary) FROM group_employees;
AVG(SALARY)
-----------
2498.571428
```

该公司员工的平均工资是 2498 元。

11．查询公司员工的工资总和——函数 SUM

数据来源于本书配套光盘\script\16-2.SQL。本例主要练习函数 SUM 的使用。SUM 函数用于求和。例如，求公司员工的工资总和：

```
SQL> SELECT SUM(salary) FROM group_employees;
SUM(SALARY)
-----------
     17490
```

12．去掉重复值（DISTINCT 子句）

数据来源于本书配套光盘\script\16-2.SQL。DISTINCT 用于去掉查询中的重复值。执行下面的 SQL 语句，将返回 3 条记录。

```
SQL> SELECT  dept_name FROM group_employees WHERE dept_name='基金管理部';
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME
----------
基金管理部
基金管理部
基金管理部
```

使用如下的 SQL 语句，将返回 1 条记录，重复数据将被压缩掉。

```
SQL> SELECT DISTINCT dept_name FROM group_employees WHERE dept_name='基金管理部';
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME
----------
基金管理部
```

13．模糊查询（LIKE）

使用选项 LIKE 和通配符百分号（%）以及下画线（_）来实现模糊查询。百分号（%）代表多个字符，下画线（_）只代表一个字符。

➢ 查询并列出所有王姓员工。

```
SQL> SELECT * FROM group_employees
  2  WHERE name LIKE  '王%';
```

SQL 语句的执行结果如下所示：

```
NAME       DEPT_NAME      SALARY
---------- ---------- ----------
王大为      基金管理部       5000
王晶        后勤部            750
王小为      基金管理部       5000
```

➢ 查询并列出所有王姓且“名”为任意一个字符的所有员工。

```
SQL> SELECT * FROM group_employees
  2  WHERE name LIKE  '王_';
```

SQL 语句的执行结果如下所示：

```
NAME       DEPT_NAME      SALARY
---------- ---------- ----------
王晶        后勤部            750
```

需要注意的是，下画线只代表一个字符，不能代表多个字符。

14．SQL 语句中的 IF 语句

在 SQL 语句中没有直接的 IF 语句，但是有两个函数（DECODE 和 CASE），它们能够实现 IF 语句的功能。当然，在 PL/SQL 应用程序开发中，有专门的 IF 语句。

📖 DECODE 函数

DECODE 函数的用法如下：

```
DECODE（条件，值 1，返回值 1，值 2，返回值 2，...值 n，返回值 n，默认值）
```

该函数的含义如下：

```
IF 条件=值 1 THEN
       RETURN（返回值 1）
ELSIF 条件=值 2 THEN
       RETURN（返回值 2）
       ......
ELSIF 条件=值 n THEN
       RETURN（返回值 n）
ELSE
       RETURN（默认值）
END IF
```

示例 1：

有一个表 inventories，这个表涉及产品和仓库的对应关系，从表中可以知道一个产品放在哪个仓库。

这个表中有如下数据：

```
PRODUCT_ID  WAREHOUSE_ID
----------  ------------
      22          1
      33          2
      22          3
      44          4
      55          6
      99         99
```

其中，PRODUCT_ID 表示产品的编号；WAREHOUSE_ID 表示仓库的编号。

执行下面的 SQL 语句：

```
SELECT product_id,
DECODE (warehouse_id, 1, 'Southlake',
2,  'San Francisco',
3,  'New Jersey',
4,  'Seattle',
'unknown')
"Location of inventory" FROM inventories;
```

得到下面的结果：

```
PRODUCT_ID  Location of inventory
----------  --------------------
      22  Southlake
      33  San Francisco
      22  New Jersey
      44  Seattle
      55  unknown
      99  unknown
```

在这个 SQL 语句中，使用了 DECODE 函数。当 warehouse_id＝1 时，返回值是'Southlake'；当 warehouse_id＝2 时，返回值是'San Francisco'；当 warehouse_id＝3 时，返回值是'New Jersey'；当 warehouse_id＝4 时，返回值是'Seattle'；当没有和 warehouse_id 相匹配的值时，返回一个默认值'unknown'。

📖 **CASE 函数**

CASE 分为简单 CASE（Simple CASE）和复杂 CASE（Searched CASE）。

➢ 简单 CASE

```
SELECT cust_last_name,
CASE credit_limit WHEN 100 THEN 'Low'
```

```
    WHEN 5000 THEN 'High'
    ELSE 'Medium' END
    FROM customers;
```

当 credit_limit 的值等于 100 时，返回值是'Low'；当 credit_limit 的值等于 5000 时，返回值是'High'，否则返回值是'Medium'。

➢ 复杂 CASE

有一个学生成绩表 ST，NAME 表示学生的名字，SCORE 表示学生的分数。

```
NAME    SCORE
jack
mary    45
tom     62
sunny   80
kite    90
```

我们将按照学生的成绩进行等级划分，如果学生分数小于 60，这个学生的成绩是 D 等；如果学生分数大于等于 60，且小于 70，这个学生的成绩是 C 等；如果学生分数大于等于 70，且小于 80，这个学生的成绩是 B 等；如果学生分数大于等于 80，这个学生的成绩是 A 等；如果学生没有分数，这个学生就没有任何等级（NO SCORE）。

执行下面的 SQL 语句，对学生成绩进行等级划分。

```
SELECT NAME, (CASE WHEN SCORE<60 THEN 'D'
       WHEN SCORE>=60 AND SCORE<70  THEN 'C'
       WHEN SCORE>=70 AND SCORE<80  THEN 'B'
       WHEN SCORE>=80 THEN 'A'
       ELSE 'NO SCORE' end )  FROM score;
```

得到如下结果：

```
NAME        GRADE
---------   ---------------
jack        NO SCORE
mary        D
tom         C
sunny       A
kite        A
```


从输出结果可以看到，学生 sunny 的分数是 80 分，因此 sunny 的成绩是 A 等；学生 jack 没有分数，因此 jack 的等级是 NO SCORE。

15．连接两个字符串

在 Oracle 中，有两种方法可以用于连接两个字符串。

➢ 用双竖线（||）连接两个字符串。||的用法说明如下。

示例 1：

```
SQL> SELECT 'BB'||'C' FROM DUAL;
'BB'||'C'
```

```
---------
BBC
```

示例 2：

```
SQL> SELECT dept_name||name FROM group_employees;
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME||NAME
--------------------
基金管理部王大为
基金管理部汪峰
后勤部许晶
```

➢ 使用 concat 函数连接两个字符串

示例：

```
SELECT concat(dept_name,name) FROM group_employees;
```

16．在查询中使用算术运算符

在 DML 语句中，可以使用加、减、乘、除等各种运算符。

➢ 在 SELECT 部分使用运算符

```
SELECT salary*1.1 FROM group_employees
```

➢ 在 WHERE 部分使用运算符

```
SELECT * FROM group_employees
WHERE salary>1000
```

17．高难度的动作——行列互换显示

在应用程序开发中，有时需要把行转换成列，有时候又需要把列转换成行，本文只是抛砖引玉地列举两个例子，希望对大家有帮助。



示例 1　把列转换成行

➢ 任务描述

有一薪水组成表 salary_composite，其数据显示如下。

```
SQL> SELECT NAME "姓名",salary_type "薪水类型",cnt "数量（元）" FROM
salary_composite;
```

SQL 语句的执行结果如下所示：

```
姓名        薪水类型              数量（元）
----------      ----------------          ----------
许晶        基本工资               1000
许晶        交通补贴                200
许晶        全勤奖                  500
```

```
刘明        基本工资              2000
刘明        全勤奖                500
刘明        效益奖                800
6 rows selected
```

接下来，要把上面的数据列换成行，互换后的数据显示如下：

```
姓名         基本工资    交通补贴     全勤奖      效益奖
---------- ---------- ---------- ---------- ----------
许晶            1000        200        500          0
刘明            2000          0        500        800
```

➢ 任务实现

执行下面的 SQL 语句实现列到行的转换。

```
SQL> SELECT NAME "姓名",
  2        SUM(decode(salary_type, '基本工资', cnt, 0)) "基本工资",
  3        SUM(decode(salary_type, '交通补贴', cnt, 0)) "交通补贴",
  4        SUM(decode(salary_type, '全勤奖', cnt, 0)) "全勤奖",
  5        SUM(decode(salary_type, '效益奖', cnt, 0)) "效益奖"
  6    FROM salary_composite
  7   GROUP BY NAME;
```

这条 SQL 语句的关键是 DECODE 函数的使用。

示例 2　把行转换成列

➢ 任务描述

示例 2 正好和示例 1 相反！

有一薪水组成表 t_salary_composite_to，表中数据显示如下：

```
SELECT name          "姓名",
       base_salary    "基本工资",
       traffic_salary "交通补贴",
       ontime_salary  "全勤奖",
       bonus         "效益奖"
 FROM t_salary_composite_to;
```


SQL 语句的执行结果如下所示：

```
姓名                 基本工资    交通补贴     全勤奖      效益奖
-------------------- ---------- ---------- ---------- ----------
许晶                       1000        200        500          0
刘明                       2000          0        500        800
```

接下来，要把上面的数据行换成列，互换后的数据显示如下：

```
姓名                薪水类型 数量（元）
------------------- -------- ----------
许晶                基本工资       1000
```

```
许晶            交通补贴        200
许晶            全勤奖          500
刘明            全勤奖          500
刘明            效益奖          800
刘明            基本工资        2000
```

➢ 任务实现

执行下面的 SQL 语句实现行到列的转换。

```
SELECT name AS "姓名", '基本工资' "薪水类型", base_salary "数量（元）"
   FROM t_salary_composite_to k
  WHERE base_salary <> 0
 UNION ALL
 SELECT NAME, '交通补贴', traffic_salary
   FROM t_salary_composite_to
  WHERE traffic_salary <> 0
 UNION ALL
 SELECT NAME, '全勤奖', ontime_salary
   FROM t_salary_composite_to
  WHERE ontime_salary <> 0
 UNION ALL
 SELECT NAME, '效益奖', bonus
   FROM t_salary_composite_to
  WHERE bonus <> 0
  ORDER BY 1 DESC;
```

这条 SQL 语句的关键是分别查询列（name、base_salary、traffic_salary、ontime_salary、bonus），然后进行集合运算。

18．如何在 SQL 语句中实现换行

函数 CHR 可以根据 ASCII 码返回字符。在 SELECT 语句中，chr(10)表示回车换行。如：

```
SQL> SELECT 'hello!'||chr(10)||'MY PUPPY !' FROM dual;
```

SQL 语句的执行结果如下所示：

```
'HELLO!'||CHR(10)||'MY
----------------------
hello!
MY PUPPY !
```

上面的例子中使用了 CHR(10)。可以看出，hello! MY PUPPY !被分成了两行。

19．随机数的产生

Oracle 专门提供一个包 DBMS_RANDOM，用于产生随机数。该包中包含一些过程或者函数，调用这些函数能够产生各种各样的随机数。

➢ 返回一个大于或等于 0 且小于 1 的随机数

```
SQL> SELECT DBMS_RANDOM.VALUE FROM dual;
```

SQL 语句的执行结果如下所示：

```
    VALUE
--------------------
0.17840927
```

➢ 返回一个大于或等于 1 且小于 50 的随机数

```
SQL> SELECT DBMS_RANDOM.VALUE(1, 50) FROM dual;
```

SQL 语句的执行结果如下所示：

```
DBMS_RANDOM.VALUE(1,50)
----------------------
  22.8586093873501
```

返回的值，可以使用 TRUNC 取整，如下所示：

```
SQL> SELECT TRUNC(DBMS_RANDOM.VALUE(1, 50)) FROM dual;
```

SQL 语句的执行结果如下所示：

```
TRUNC(DBMS_RANDOM.VALUE(1,50))
------------------------------
         15
```

➢ 返回一个指定长度的随机字符串

```
SQL> SELECT DBMS_RANDOM.STRING('A',20) FROM dual;
```

SQL 语句的执行结果如下所示：

```
DBMS_RANDOM.STRING('A',20)
--------------------------------------------------
UQnAqvCoNYTudrGAwFqy
```

本例返回长度是 20 的字符串。

DBMS_RANDOM.STRING 的格式是 DBMS_RANDOM.STRING (opt,len)，其中 opt 取值如下：

- ✓ 'u'或'U'　返回大写字母
- ✓ 'l'或'L'　返回小写字母
- ✓ 'a'或'A'　大小写字母混合
- ✓ 'x'或'X'　大写字母和数字混合
- ✓ 'p'或'P'　任意可显示字符

➢ 随机返回一个整数

```
SQL> SELECT dbms_random.random FROM dual;
```

SQL 语句的执行结果如下所示：

```
RANDOM
```

```
----------------------
 -40216577
```

20．区间查询 BETWEEN...AND

BETWEEN...AND 用于测试一个值是否位于某个区间，BETWEEN...AND 的语法如下所示：

```
expr1 BETWEEN expr2 AND expr3
```

测试表达式 expr1 的值是否位于 expr2 与 expr3 之间。expr1、expr2、expr3 可以是数字型、字符型、日期型，但是三者的数据类型必须相同。

➢ 测试数字 1.5 是否介于 1 到 3 之间

```
SQL> SELECT '测试数字' FROM dual WHERE 1.5 BETWEEN 1 AND 3;
'测试数字'
----------
测试数字
```

➢ 测试字符 c 是否介于 a 到 d 之间

```
SQL> SELECT '测试字符' FROM dual WHERE 'c' BETWEEN 'a' AND 'd';
'测试字符'
----------
测试字符
```

➢ 测试日期 2009 年 9 月 9 日是否介于 2008 年 8 月 8 日到当前系统时间之间

```
SQL> SELECT '测试日期和时间'
  2    FROM dual
  3   WHERE to_date('2009-09-9', 'YYYY-MM-DD') BETWEEN
  4         to_date('2008-08-8', 'YYYY-MM-DD') AND SYSDATE;
'测试日期和时间'
----------------
测试日期和时间
```

相反，我们可以使用 NOT BETWEEN...AND 测试某个值是否不介于指定区间。

21．空值（NULL）的使用

需要注意的是，空字符和空值（NULL）是两回事。NULL 表示值未知，空字符表示值已知，只是值为空字符串而已。

➢ 创建测试表 T_NULL

```
CREATE TABLE T_NULL
(name VARCHAR2(20),
dept_name VARCHAR2(20));
```

➢ 往表 T_NULL 中插入数据，dept_name 设置成 NULL

```
INSERT INTO T_NULL(name,dept_name)
```

```
VALUES('马大帅','');

INSERT INTO T_NULL(name,dept_name)
VALUES('马小帅',NULL);
```

特别注意的是，上面两个 SQL 语句实现的是相同的功能。

- 往表 T_NULL 中插入数据，dept_name 是空字符串而不是空值（NULL）

```
INSERT INTO T_NULL(name,dept_name)
VALUES('马老帅',' ');
COMMIT;
```

- 查出部门为空值（NULL）的人的信息

```
SQL> SELECT * FROM T_NULL WHERE dept_name IS NULL;
```

SQL 语句的执行结果如下所示：

```
NAME               DEPT_NAME
------------------- --------------------
马大帅
马小帅
```

IS NULL 判断是否为空值（NULL）。

- 查出部门不为空值（NULL）的人的信息

```
SQL> SELECT * FROM T_NULL WHERE dept_name IS NOT NULL;
```

SQL 语句的执行结果如下所示：

```
NAME               DEPT_NAME
------------------- --------------------
马老帅
```

22．IN 与 NOT IN 的陷阱

判断条件 IN 用于测试一个值是否位于一个列表之中。

- 测试 1 是否在列表（1, NULL）中

```
SQL> SELECT 'TRUE' FROM dual WHERE 1 IN (1, NULL);
'TRUE'
------
TRUE
```

查询结果是 TRUE，因此 1 在列表（1, NULL）中。

- 测试员工的部门编号 dept_id 是否在部门信息表（dept_infor）中

```
SELECT * FROM employees
WHERE dept_id IN
(SELECT dept_id  FROM dept_infor)
```

- NOT IN

```
SQL> SELECT 'TRUE' FROM dual WHERE 1 NOT IN (1,2, NULL);
'TRUE'
------
```

此条 SQL 语句没有返回值，因为 1 在给定的列表中。

➢ 空值（NULL）陷阱

```
SQL> SELECT 'TRUE' FROM dual WHERE 1 NOT IN (2, NULL);
'TRUE'
------
```

此条 SQL 语句也没有返回值，这是“空值陷阱”造成的。通常我们认为 1 肯定不在列表（2, NULL）中，理应返回“TRUE”值，但是 Oracle 的判断规则是：对于 NOT IN，如果测试值不在列表中，且列表中有一个值为 NULL，则测试结果是 FALSE。

23．EXISTS 与 NOT EXISTS

EXISTS 用于测试数据行的存在性，EXISTS 的语法如下：

```
EXISTS (subquery)
```

如果子查询返回至少一行，则测试值是 TRUE。请看下面的例子：

```
SELECT *
FROM student S
WHERE EXISTS
(SELECT * FROM dept_infor E
WHERE S.department_id
= E.department_id)
```

本例中，如果院系信息表（dept_infor）中**存在**一条记录（该记录的 department_id 与学生信息表 student 中的 department_id 相同），则该学生的信息将被返回。

也可使用 NOT EXISTS 判断不存在的记录。

24．记录为空时也返回一条默认记录

有时候，如果查询没有记录返回，我们也要求该 SQL 语句返回一条默认记录，默认记录也许在我们的应用程序中有不同的用途。

```
SELECT NAME, dept_name
  FROM t_null
UNION
SELECT '马晓曹', '未知' FROM dual WHERE NOT EXISTS (SELECT * FROM t_null)
```

如果表中有记录，将返回表中的记录；如果表中不存在记录，将返回如下默认记录：

```
NAME    DEPT_NAME
马晓曹    未知
```

25．将查询结果导成 XML

利用 SELECT 语句，结合客户端工具 PL/SQL Developer，可以将查询结果导成 XML。这种方法加入了手工干预（相信大家还会有更好的办法）。将查询结果导成 XML 的示例如下。

STEP 01 使用 PL/SQL Developer 登录数据库。

STEP 02 单击菜单“File”→“New”→“SQL Window”，打开一个窗口。

STEP 03 在该窗口中执行下面的查询语句，将会见到如图 16-3 所示的查询结果。

```
SELECT * FROM phone_infor
```

	PHONE_NAME	PRODUCER	PRICE
1	N97	诺基亚	700
2	E66	诺基亚	300
3	E67	诺基亚	400

图 16-3　查询结果

STEP 04 选中所有数据，右键单击上图，将弹出一个菜单，选择“Export Resuts”→“XML File”，弹出一窗口，在此需要指定 XML 文件的文件名，输入文件名，单击“保存”按钮，XML 文件成功产生。

26．找出表中的重复记录

有一表中数据关于 ID 重复，表中数据如下：

```
SQL> SELECT * FROM t_doublerow;
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
111        小明
231        小华
111        小东
```

找出表中的重复记录：

```
SELECT * FROM  t_doublerow
WHERE ID IN(
SELECT ID FROM t_doublerow
GROUP BY ID
HAVING(COUNT(ID)>=2)
);
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
111        小明
111        小东
```

子查询找出有重复值的 ID，主查询根据 ID 找到 ID 重复的记录。

27．分组统计（GROUP BY 子句）

本节将以员工工资表 group_employees 为基础进行分组统计。员工工资表含有以下数据（创建脚本，请参考本书配套光盘\script\16-2.SQL）。

```
NAME            DEPT_NAME     SALARY
----------      ----------  ----------
王大为          基金管理部      5000
汪峰            基金管理部      4000
许晶            后勤部           800
王晶            后勤部           750
刘晓            后勤部           740
李俊            保安部          1200
王小为          基金管理部      5000
```

其中，NAME表示员工姓名；DEPT_NAME 表示员工所在部门；SALARY 表示员工工资。

以下的示例将以该表为基础。

🕮 列出每个部门的员工人数（COUNT 函数的使用）

统计每个部门的员工人数：

```
SQL> SELECT dept_name,count(*)
  2  FROM group_employees
  3  GROUP BY dept_name
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME    COUNT(*)
---------- ----------
保安部              1
后勤部              3
基金管理部          3
```

GROUP BY 表示按照分组进行统计，本例中，按照部门（dept_name）进行分组。count(*)统计的是每个部门的员工人数。

从查询结果可知：保安部有 1 人；后勤部有 3 人；基金管理部有 3 人。

🕮 找出每个部门工资最高的员工（MAX 函数的使用）

按照部门进行分组，找出每个部门工资最高的人。

```
SELECT dept_name,name,salary
FROM group_employees A
WHERE (A.dept_name,A.salary) IN(
SELECT dept_name,MAX(salary)
FROM group_employees B
```

```
    GROUP BY dept_name)
    ORDER BY dept_name
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME          NAME          SALARY
----------      ----------       ----------
保安部             李俊            1200
后勤部             许晶             800
基金管理部         王大为           5000
基金管理部         王小为           5000
```

在上面的 SQL 语句中，粗体部分用于查出每个部门及每个部门的最高工资，然后根据每个部门及最高工资找出每个部门工资最高的人。MAX 函数用于计算每个部门的最高工资。

本例中，保安部工资最高的员工是李俊，工资是 1200 元。

列出每个部门工资前 *n* 高的所有员工

本节的目标是找出每个部门工资第 1 高、第 2 高…第 *n* 高的所有员工。如市场部的最高工资是 8000 元，次高工资是 7000 元，第三高是 5000 元…依此类推。如果想找出工资排在前 3 名的员工，则工资是 8000 元、7000 元、5000 元的所有员工将被列出来。

下面的例子将列出每个部门工资排在前两名的所有员工的信息。

STEP 01 打开 SQL*Plus，修改 SQL*Plus 的环境变量。

执行 SQL*Plus 命令 SET SERVEROUTPUT ON。

STEP 02 执行下面的 SQL 语句，将列出每个部门工资排在前两名的所有员工（topn:=2）。

```
DECLARE
        v_dept_name varchar2(20);--部门名字
        v_name varchar2(20);--员工姓名
        v_salary number; --员工工资
        v_plsql_block VARCHAR2(1000);--存放动态 SQL 语句
        topn number ;--指示前 n 高工资
        cursor      v_c1(n  number)  IS  SELECT  dept_name  ,name,salary  FROM
admin_work_area3 order by dept_name;
BEGIN
        topn:=2;--本例中只列出工资排在前两名的所有员工的信息
         v_plsql_block := 'TRUNCATE TABLE admin_work_area3';
          EXECUTE IMMEDIATE v_plsql_block;--删除临时表中的数据

        v_plsql_block := 'DROP TABLE admin_work_area3';
          EXECUTE IMMEDIATE v_plsql_block;  --如果临时表存在，删除临时表

         v_plsql_block  :=  'CREATE  GLOBAL  TEMPORARY  TABLE  admin_work_area3
(dept_name varchar2(10),name varchar2(10),salary number) ON COMMIT PRESERVE
ROWS';
```

```
        EXECUTE IMMEDIATE v_plsql_block; --创建临时表，该临时表用于临时存放工资前n高的员工信息

        FOR i IN 1..topn LOOP—获得工资前n高的员工信息，并存入临时表
          INSERT INTO admin_work_area3
            SELECT dept_name ,name,salary
            FROM group_employees A
            WHERE (A.dept_name,A.SALARY) IN(
              SELECT dept_name,MAX(SALARY)
              FROM group_employees B
              WHERE (name) not in(SELECT name FROM admin_work_area3)
              GROUP BY dept_name);
        END LOOP;

        OPEN v_c1(topn);
        DBMS_OUTPUT.PUT_LINE('部门        '||'姓名         '||'   工资');

        LOOP—列出临时表的内容
            FETCH v_c1 INTO v_dept_name,v_name,v_salary;
            EXIT WHEN v_c1%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(v_dept_name||'  '||v_name||'  '||v_salary);
        END LOOP;

END;
```

找出每个部门工资最低的员工（MIN 函数的使用）

按照部门进行分组，找出每个部门工资最低的人。

```
SELECT dept_name,name,salary
FROM group_employees A
WHERE (A.dept_name,A.salary) IN(
    SELECT dept_name,MIN(salary)
    FROM group_employees B
    GROUP BY dept_name)
ORDER BY dept_name
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME  NAME          SALARY
----------      ----------      ----------
保安部        李俊             1200
后勤部        刘晓             740
基金管理部    汪峰             4000
```

使用 MIN 函数计算每个部门的最低工资。

本例中，后勤部工资最低的人是刘晓，工资是 740 元。

算出每个部门的平均工资（AVG 函数的使用）

使用 AVG 函数计算平均值，本例计算每个部门的平均工资。

```
SELECT dept_name ,AVG(salary)
FROM group_employees A
GROUP BY dept_name
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME  AVG(SALARY)
----------  -----------
保安部           1200
后勤部      763.333333
基金管理部  4666.66667
```

找出平均工资最高的部门

```
SELECT *
FROM(--计算每个部门的平均工资
SELECT dept_name ,AVG(salary)
FROM group_employees
GROUP BY dept_name
ORDER BY dept_name DESC
)
WHERE ROWNUM<=1;--取排序中的第一行
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME  AVG(SALARY)
---------- -----------
基金管理部 4666.666666
```

从上述结果可以看出，平均工资最高的部门是基金管理部。

找出平均工资最低的部门

```
SELECT *
FROM(--计算每个部门的平均工资
SELECT dept_name ,AVG(salary)
FROM group_employees
GROUP BY dept_name
ORDER BY dept_name ASC
)
WHERE ROWNUM<=1;--取排序中的第一行
```

SQL 语句的执行结果如下所示：

```
DEPT_NAME  AVG(SALARY)
---------- -----------
保安部           1200
```

从上述结果可以看出，平均工资最低的部门是保安部。

分级汇总（ROLLUP）

所谓“分级汇总”，就是一级一级地对数据进行汇总。

有一鞋厂的销售数据如下：

```
DEAL_NAME            MONEY      DEAL_DATE
-------------------- ---------- -----------
足球鞋                  20000        2010-1-31
登山鞋                  20000        2011-12-3
旱冰鞋                   1000        2009-5-10
雨鞋                     1020        2009-5-20
运动鞋                   2020        2009-8-1
篮球鞋                   8000        2009-8-15
雨鞋                    20000        2009-12-31
```

其中，DEAL_NAME 表示销售的鞋的名称；MONEY 表示销售金额；DEAL_DATE 表示销售日期。

执行下面的 SQL 语句，对销售数据进行分级汇总。

```
    SELECT TO_CHAR(deal_date, 'YYYY') 年,TO_CHAR(deal_date,'MM') 月,SUM(money)
销售收入
    FROM shoe_sales
    GROUP BY ROLLUP(TO_CHAR(deal_date, 'YYYY'),TO_CHAR(deal_date,'MM'))
```

SQL 语句执行的结果如下所示：

```
年          月          销售收入
2009        05          2020
2009        08          10020
2009        12          20000
2009                    32040
2010        01          20000
2010                    20000
2011        12          20000
2011                    20000
                        72040
```

通过 GROUP BY ROLLUP 实现分级汇总，本例中，先按照月（TO_CHAR(deal_date,'MM')）进行汇总，接着再按照年（TO_CHAR(deal_date, 'YYYY')）进行汇总，最后进行综合汇总（或者叫总的汇总），值 72040 是综合汇总。所以 ROLLUP 实现的是层层汇总。

多维汇总（CUBE）

多维汇总通过 GROUP BY CUBE 来实现，所谓“多维汇总”就是从多个角度（维度）对同一数据进行汇总。下面，我们将通过一个例子讲解多维汇总，本例的数据来源同分级汇总（ROLLUP）的数据来源。执行下面的 SQL 语句：

```
    SELECT TO_CHAR(deal_date, 'YYYY') 年,TO_CHAR(deal_date,'MM') 月,SUM(money)
销售收入
```

```
FROM shoe_sales
GROUP BY CUBE(TO_CHAR(deal_date, 'YYYY'),TO_CHAR(deal_date,'MM'))
```

SQL 语句执行的结果如下所示：

年	月	销售收入
		72040
	01	20000
	05	2020
	08	10020
	12	40000
2009		32040
2009	05	2020
2009	08	10020
2009	12	20000
2010		20000
2010	01	20000
2011		20000
2011	12	20000

本例从年（TO_CHAR（deal_date, 'YYYY'））和月（TO_CHAR(deal_date,'MM')）两个角度进行汇总。值 72040 是综合汇总（或者叫总的汇总）。

28．字符处理

本节重点介绍字符函数的使用技巧。由于字符函数众多，我们不可能逐一介绍，还请大家谅解。对于本书没有涉及的函数，请大家参考 Oracle 的官方文档。

批量替换特殊字符

有两个函数 REPLACE 和 TRANSLATE，都可以用于字符的批量替换，唯一不同的是 REPLACE 的单位是**字符串**，TRANSLATE 的单位是**字符**。

- 整体替换——REPLACE 函数

```
SQL> SELECT REPLACE('ckna is great and i am FROM ckna', 'ck', 'Chi')
  2   FROM dual;
```

SQL 语句的执行结果如下所示：

```
REPLACE('CKNAISGREATANDIAMFROM
--------------------------------
China is great and i am FROM China
```

替换'ckna is great and i am FROM ckna'中的字符，字符串“ck”被字符串“Chi”整体替换。

➢ 逐个字符替换——TRANSLATE 函数

```
SQL> SELECT TRANSLATE('rbfecyp', 'bec', '123') FROM dual;
```

SQL 语句的执行结果如下所示：

```
TRANSLATE('RBFECYP','BEC','123
------------------------------
r1f23yp
```

替换'rbfecyp'中的字符，字符 b 被 1 替换，字符 e 被 2 替换，字符 c 被 3 替换，它们之间是一一对应的关系。这里替换的单位是字符，而不是字符串，这是与函数 REPLACE 截然不同的地方。TRANSLATE 函数的语法是：

TRANSLATE(string1, string2, string3)

string2 与 string3 中的字符一一对应，如图 16-4 所示。

需要注意的是，如果一个字符同时存在于 string1 及 string2 中，而不存在于 string3 中，则该字符将被从字符串（string1）中删除，请看下面的例子：

```
SQL> SELECT TRANSLATE ('rbfecyp', 'bec', '#') FROM dual;
```

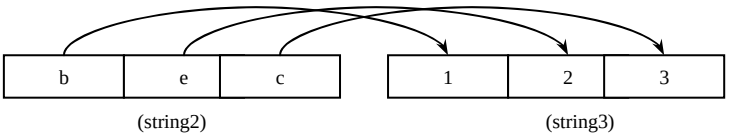


图 16-4　TRANSLATE 函数参数

SQL 语句的执行结果如下所示：

```
TRANSLATE('RBFECYP','BEC','#')
------------------------------
r#fyp
```

本例中，e 和 c 同时存在于 string1 及 string2 中，而不存在于 string3 中，则 e 和 c 从最终结果中被删除。

去除字符串中的数字

利用字符处理 TRANSLATE 函数，可以去除字符串中的数字。有关 TRANSLATE 函数的用法，**请参考批量替换特殊字符的内容。**去掉字符串中的数字举例如下：

```
SQL> SELECT translate('m8y 3d4e5al', '#1234567890', '#')
```

```
  2  FROM dual;
```

SQL 语句的执行结果如下所示：

```
TRANSLATE('M8Y3D4E5AL','#12345
------------------------------
my deal
```

📖 统计字符串中某个字符（字符串）出现的次数

➢ 统计某个**字符**在字符串中出现的次数

参考 TRANSLATE 函数的用法，我们可以计算一个字符（而不是字符串）在字符串中出现的次数。如计算字符 b 在字符串'bbcyyddkbbc'中出现的次数。

```
SELECT length(translate('bbcyyddkbbc', 'b' || 'bbcyyddk', 'b')) FROM dual;
```

SQL 语句的执行结果如下所示：

```
LENGTH(TRANSLATE('BBCYYDDKBBC'
------------------------------
                             4
```

字符 b 在字符串'bbcyyddkbbc'中出现 4 次。

有关 TRANSLATE 函数的用法，**请参考批量替换特殊字符的内容。**

➢ 统计某个**字符串**在另一字符串中出现的次数

```
SELECT (length(string1) - length(REPLACE(string1, string2))) /
       length(string2)
  FROM dual
```

其中，string1 表示将在该字符串中进行搜索；string2 表示查询该字符串在 string1 出现的次数；REPLACE(string1, string2)表示从字符串 string1 中去除字符串 string2 的值。

例：

```
SELECT (length('ckna is great and i am FROM ckna') -
       length(REPLACE('ckna is great and i am FROM ckna', 'ck'))) /
       length('ck')
  FROM dual
```


SQL 语句的执行结果如下所示：

```
(LENGTH('CKNAISGREATANDIAMFROM
------------------------------
             2
```

从查询结果可以看出，字符串 ck 在字符串'**ck**na is great and i am FROM **ck**na'中出现了两次。

📖 判断一个字符串是否全部为数字

通过 Oracle 自带的函数 TRANSLATE 和用户自定义函数 veryfy_num，都可以判断一个字符串是否全部为数字。TRANSLATE 函数有诸多限制，并且该函数也是间接判断一个

字符串是否全部为数字，鉴于此，笔者推荐大家使用我们自定义的函数 veryfy_num。

➢ 如果字符串全为数字，返回 Y

```
SQL> SELECT veryfy_num('12') FROM dual;
```

SQL 语句的执行结果如下所示：

```
VERYFY_NUM('12')
----------------------
Y
```

➢ 如果字符串不全为数字，返回 N

```
SQL> SELECT veryfy_num('f12') FROM dual;
```

SQL 语句的执行结果如下所示：

```
VERYFY_NUM('F12')
----------------------
N
```

➢ 创建函数 veryfy_num

```
CREATE OR REPLACE FUNCTION veryfy_num(par IN VARCHAR2) RETURN VARCHAR2 IS
  v_tem NUMBER;
BEGIN
  BEGIN
    v_tem := to_number(par);
    RETURN('Y');
  EXCEPTION
    WHEN OTHERS THEN
     RETURN('N');
  END;
END veryfy_num;
```

去掉字符串中的空格

函数 REPLACE 和 TRANSLATE 都可以帮助我们去掉字符串中的空格。

➢ 利用 REPLACE 函数去掉字符串中的空格

例：去掉字符串'so me th ing'中的空格。

```
SQL> SELECT REPLACE('so me th ing', ' ', '')
  2  FROM dual;
```

SQL 语句的执行结果如下所示：

```
REPLACE('SOMETHING','','')
--------------------------
something
```

第二个参数是一个空字符串。

➢ 利用 TRANSLATE 函数去掉字符串中的空格

例：去掉字符串'so me th ing'中的空格。

```
SQL> SELECT TRANSLATE ('so me th ing', '# ', '#') FROM dual;
```

SQL 语句的执行结果如下所示：

```
TRANSLATE('SOMETHING','#','#')
------------------------------
something
```

由于印刷的关系，看不清 TRANSLATE 函数第二个参数的真实值，第二个参数的井号（#）之后其实还有一个空格。

🕮 删除字符串的前几个字符

利用 SUBSTR 函数可以删除字符串的前几个字符。

示例 1：删除字符串“xxxLLL”的前两个字符

```
SQL> SELECT SUBSTR ('xxxLLL', 3) FROM dual;
```

SQL 语句的执行结果如下所示：

```
SUBSTR('XXXLLL',3)
------------------
xLLL
```

示例 2：删除字符串“测试日期和时间”的前 3 个字符

```
SQL> SELECT SUBSTR ('测试日期和时间', 4) FROM dual;
```

SQL 语句的执行结果如下所示：

```
SUBSTR('测试日期和时间',4)
--------------------------
日期和时间
```

SUBSTR 函数以字符为单位计算位置。函数 SUBSTRB 的功能和 SUBSTR 类似，只是在计数时，它以字节为单位。

在使用 SUBSTR 及 SUBSTRB 函数时要格外小心，在计算字符位置时，理解上会出现小小的偏差，因此，在生产系统应用这两个函数时（尤其涉及数据修改），请先在实验环境中进行测试。

🕮 字母大小写的转换

➢ LOWER 函数用于将字符串转换成小写

```
SQL> SELECT LOWER('MY dog!') FROM dual;
```

SQL 语句的执行结果如下所示：

```
LOWER('MYDOG!')
---------------
my dog!
```

➢ UPPER 函数用于将字符串转换成大写

```
SQL> SELECT UPPER('MY dog!') FROM dual;
```

SQL 语句的执行结果如下所示：

```
UPPER('MYDOG!')
---------------
MY DOG!
```

➢ INITCAP 函数用于将字符串中每个单词的首字母转换成大写，其他字母小写

```
SQL> SELECT INITCAP('my doG!') FROM dual;
```

SQL 语句的执行结果如下所示：

```
INITCAP('MYDOG!')
-----------------
My Dog!
```

🕮 **汉字转拼音**

进行软件开发时，我们需要根据给出的汉字，得到汉字的拼音。汉语转拼音的过程其实并不难，有诸多方法可供选择，有只利用函数的，有利用外部表的，有利用第三方工具的。本节将讲述如何利用外部表实现汉字到拼音的转换。利用外部表实现汉字到拼音的转换需要一个存放汉字拼音对照关系的文件 hanzipinyin.txt（在本书配套光盘的\script 下可以找到该文件）。汉字转拼音的脚本放在本书配套光盘的\script 下脚本文件 turn_to_pinyin.sql 中。

下面我们将讲述汉字转拼音的全部过程。

STEP 01 创建目录。

在操作系统中创建目录 D:\pinyin\data、D:\pinyin\log、D:\pinyin\bad。Oracle 对这些目录要有读写权限。

STEP 02 以数据库管理员的身份登录数据库。

STEP 03 创建目录对象。

```
CREATE OR REPLACE DIRECTORY dat_dir_p
AS 'D:\pinyin\data';
CREATE OR REPLACE DIRECTORY log_dir_p
AS 'D:\pinyin\log';
CREATE OR REPLACE DIRECTORY bad_dir_p
AS 'D:\pinyin\bad';
```

这里的目录对象是一个数据库对象，目录对象和实际的操作系统目录有关系，如目录对象 dat_dir_p 对应操作系统目录 D:\pinyin\data。D:\pinyin\data 存放数据，D:\ pinyin \log 存放日志，D:\ pinyin \bad 存放坏文件。如果一条记录格式不对或者该记录引起数据库错误，这条记录将被写到坏文件中。

STEP 04 授予 Oracle 用户对这些目录的读写权限。

```
GRANT READ ON DIRECTORY dat_dir_p TO itme;
GRANT READ,WRITE ON DIRECTORY log_dir_p TO itme;
GRANT READ,WRITE ON DIRECTORY bad_dir_p TO itme;
```

本例授予数据库用户 itme 对这些目录将有读或者写的权限。这里的读写概念是基于数据库层面的。为了使用外部表，Oracle 要求：

- 用户对存放数据的目录要有读的权限；
- 用户对存放日志文件的目录要有写的权限；
- 用户对存放坏文件的目录要有写的权限。

STEP 05 复制文件。

将光盘上\script 下的文件 hanzipinyin.txt 复制到目录 D:\pinyin\data 中。特别注意的是，外部表的数据不是存放在数据库中（数据库仅仅存放外部表的定义），而是存放在操作系统文件 hanzipinyin.txt 中。

STEP 06 以用户 itme 的身份登录数据库。

```
CONNECT itme;
```

STEP 07 创建外部表 tb_hanzipinyin。

创建外部表 tb_hanzipinyin 的 SQL 语句如下所示：

```
CREATE TABLE tb_hanzipinyin
(hangzi VARCHAR2(10),
pinyin VARCHAR2(20)
)
ORGANIZATION EXTERNAL
(
TYPE ORACLE_LOADER
DEFAULT DIRECTORY dat_dir_p
ACCESS PARAMETERS
(
RECORDS delimited BY NEWLINE
BADFILE bad_dir_p:'empxt%a_%p.bad'
LOGFILE log_dir_p:'empxt%a_%p.log'
FIELDS TERMINATED BY '#'
MISSING FIELD VALUES are NULL
(hangzi,pinyin)

)
LOCATION ('hanzipinyin.txt')
)
PARALLEL
REJECT LIMIT UNLIMITED;
```

有关创建外部表的语法，**请参考本书 16.4.1 节的内容**，在此，我们只是对这段 SQL 语句进行简要描述。

- hangzi 是字段，表示汉字。
- pinyin 是字段，表示汉字的拼音（一个汉字可能有多个发音）。
- FIELDS TERMINATED BY '#'表示字段之间以井号分割，这里的说明和文件

hanzipinyin.txt 中的数据格式要对应（你可以打开该文件查看）。

STEP 08 根据汉字查询拼音。

```
SQL> SELECT * FROM tb_hanzipinyin WHERE hangzi LIKE '参';
```

SQL 语句的执行结果如下所示：

```
HANGZI    PINYIN
---------- --------------------
参        can cen shen
```

从输出可以看出，汉字“参”有三个读音。

STEP 09 创建函数 hanzi_to_pinyin。

函数的输入值是汉字，输出值是该汉字的拼音，如果出现其他错误，则会返回信息“ERROR”。函数创建的 SQL 语句如下所示：

```
CREATE OR REPLACE FUNCTION hanzi_to_pinyin(p_hanzi in VARCHAR2)
  RETURN VARCHAR2 IS
  v_return_pinyin VARCHAR2(20);
  BEGIN
    SELECT pinyin
      INTO v_return_pinyin
      FROM tb_hanzipinyin
     WHERE hangzi = p_hanzi;
  RETURN        v_return_pinyin;
  EXCEPTION      WHEN OTHERS THEN RETURN 'ERROR';
END;
```

STEP 10 利用函数返回汉字的拼音。

```
SQL> SELECT hanzi_to_pinyin('小') FROM DUAL;
```

SQL 语句的执行结果如下所示：

```
HANZI_TO_PINYIN('小')
--------------------------------------------------------------------
xiao
```

截取字符串里某字符之前的字符串

例：截取字符串'come on baby#stttttttttt'中井号（#）之前的字符串。

```
SQL> SELECT SUBSTR('come on baby#stttttttttt', 1, INSTR('come on
baby#stttttttttt', '#') - 1) FROM DUAL;
```

SQL 语句的执行结果如下所示：

```
SUBSTR('COMEONBABY#STTTTTTTTTT
------------------------------
come on baby
```

人民币大小写转换

在银行业务处理中，经常需要把小写人民币翻译成大写。我们设计了一个函数，该函数能够很好地完成该任务。该函数的演示如下：

```
SQL> SELECT RMB_daxie(4340789061250)  FROM dual;
```

SQL 语句的执行结果如下所示：

```
RMB_DAXIE(4340789061250)
----------------------------------------------------
肆万叁仟肆佰零柒亿捌仟玖佰零陆万壹仟贰佰伍拾元
```

创建函数 RMB_daxie 的脚本在光盘\script 下的文件 RMB_daxie.SQL 中，在创建该函数之前，需要创建其他配套函数，这些函数都是我们自定义的函数。详细过程说明如下。

STEP 01 创建函数 arabic_daxie()，该函数把阿拉伯数字转换成大写。

STEP 02 创建函数 fenduanchuli()，分段处理函数，该函数把小写人民币分成亿、万、元 3 段进行处理。

STEP 03 创建函数 rmb_daxie()，这是人民币大写的总函数。



注 1：函数 RMB_daxie 花费笔者较长的时间进行调试，如有转载，请注明出处。

注 2：函数 RMB_daxie 需要更长时间的测试，如若用于生产环境，请测试后再进行部署。

29．期时间处理

本节将要介绍日期时间的相关处理，并演示处理中的一些技巧。

获得系统的当前日期和时间（SYSDATE）

通过 SYSDATE 函数和哑表 DUAL 的联合使用，可以获得系统的日期和时间。

- 获得系统的日期和时间

```
SQL> SELECT SYSDATE FROM DUAL;
```

SQL 语句的执行结果如下所示：

```
SYSDATE
-----------
2009-9-14
```

该查询是显示日期还是显示时间或者两者同时显示，主要依靠初始化参数 NLS_DATE_FORMAT 的设置。

- 只显示日期

```
SQL> SELECT TO_CHAR (SYSDATE, 'MM-DD-YYYY') "NOW"
  2  FROM DUAL;
```

SQL 语句的执行结果如下所示：

```
NOW
----------
```

```
09-14-2009
```

➢ 只显示时间

```
SQL> SELECT TO_CHAR (SYSDATE, 'HH24:MI:SS') "NOW"
  2  FROM DUAL;
```

SQL 语句的执行结果如下所示：

```
NOW
--------
10:20:45
```

➢ 日期和时间都显示。

```
SQL> SELECT TO_CHAR (SYSDATE, 'MM-DD-YYYY HH24:MI:SS') "NOW"
  2  FROM DUAL;
```

SQL 语句的执行结果如下所示：

```
NOW
-------------------
09-14-2009 10:21:02
```

其中，MM-DD-YYYY HH24:MI:SS 是掩码，用于指定日期时间的显示格式。

修改日期的显示格式

在 SQL*Plus 查询系统时间时，时间（日期）的显示比较凌乱或者不符合东方人的阅读习惯，我们可以修改初始化参数 NLS_DATE_FORMAT 来个性化时间（日期）的显示格式。NLS_DATE_FORMAT 控制时间（日期）的显示格式。

STEP 01 查询当前的日期显示格式，如下所示：

```
SQL> SELECT sysdate FROM dual;
```

SQL 语句的执行结果如下所示：

```
SYSDATE
--------------
15-9月 -09
```

STEP 02 修改日期的显示格式。

```
ALTER SESSION SET nls_date_format='YYYY-MM-DD HH24:MI:SS';
```

其中，YYYY 表示年（如 2006 年）；MM 表示月份（如 04 表示 4 月份）；HH 表示小时，HH24 表示按照 24 小时计算的时间；MI 表示分钟；SS 表示秒。

STEP 03 查看日期的显示格式是否发生了改变。

```
SQL> SELECT sysdate FROM dual;
```

SQL 语句的执行结果如下所示：

```
SYSDATE
-------------------
```

```
2009-09-15 16:37:05
```

输出显示，当前的系统日期是 2009 年 9 月 15 日，时间为 16 点 37 分 5 秒，说明日期的显示格式已经发生了变化。

ALTER SESSION SET nls_date_format 所作的修改，只对本次会话（本次登录）有效。如果用户重新登录数据库，刚才的修改将会失效。如果要想一劳永逸，需要修改初始化文件里的 NLS_DATE_FORMAT 参数。

求某个月有多少天

利用 LAST_DAY 函数可以求出一个月有多少天。

```
SQL> SELECT SYSDATE,TO_NUMBER(TO_CHAR(LAST_DAY(SYSDATE),'DD')) FROM DUAL;
```

SQL 语句的执行结果如下所示：

```
SYSDATE    TO_NUMBER(TO_CHAR(LAST_DAY(SYS
---------- ------------------------------
2009-9-16                            30
```

从查询结果可以看出，2009 年 9 月共有 30 天。

其中，SYSDATE 是当前系统时间；DD 是日期掩码，表示号数；TO_NUMBER 把字符型转换成数字型。

计算一个月还剩多少天

利用 LAST_DAY 函数可以求出一个月还剩多少天。

```
SQL> SELECT SYSDATE,LAST_DAY(SYSDATE) - SYSDATE "Days Left"
  2  FROM DUAL;
```

SQL 语句的执行结果如下所示：

```
SYSDATE     Days Left
----------- ----------
2009-9-16           14
```

今天是 2009 年 9 月 16 日，这个月还剩 14 天。

LAST_DAY(SYSDATE)得到一个月中最后一天的日期；SYSDATE 是当前系统日期。

今天星期几

利用日期掩码“DAY”可以知道今天是星期几。

```
SQL> SELECT TO_CHAR(SYSDATE,'DAY') FROM dual;
```

SQL 语句的执行结果如下所示：

```
TO_CHAR(SYSDATE,'DAY')
----------------------
星期三
```

当然，也可以知道任意日期是星期几。

📖 今天是几号

通过 EXTRACT 函数，可以获知今天是几号。

```
SQL> SELECT EXTRACT(DAY FROM SYSDATE) "今天几号" FROM dual;
```

SQL 语句的执行结果如下所示：

```
今天几号
----------
16
```

📖 今天是一年的第几天

利用日期掩码“DDD”可以知道今天是一年的第几天。

```
SQL> SELECT TO_CHAR(SYSDATE,'DDD') FROM dual;
```

SQL 语句的执行结果如下所示：

```
TO_CHAR(SYSDATE,'DDD')
----------------------
259
```

30．集合运算

集合运算包括 UNION、UNION ALL、INTERSECT 和 MINUS。本节例子涉及的数据请参考光盘上的\script\16-6.SQL。

表 product_A 中的数据如下：

```
ID         NAME
---------- ----------
425        mp3
425        落地音响
444        牛奶杯
```

表 product_B 中的数据如下：

```
ID         NAME
---------- ----------
425        mp3
555        台灯
```

其中，ID 表示产品编号；NAME 表示产品名字。

📖 合并两个查询（UNION ALL 与 UNION 的区别）

UNION 用于合并两个查询（两个结果集）。

■ UNION ALL

```
SQL> SELECT id,name FROM product_A
  2  UNION ALL
  3  SELECT id,name FROM product_B;
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425        mp3
425        落地音响
444        牛奶杯
425        mp3
555        台灯
```

UNION ALL 将列出两个集合中的所有记录，无论它们是否重复。两条记录是否重复的判断标准为：一条记录中的每个值和另一条记录中相应的值相等，如记录 AA BB 和记录 AA BB 相同，但记录 AA BB 和记录 AA CC 不相同。

■ UNION

```
SQL> SELECT id,name FROM product_A
  2  UNION
  3  SELECT id,name FROM product_B;
```

SQL 语句的执行结果如下所示：

```
 ID         NAME
---------- ----------
425        mp3
425        落地音响
444        牛奶杯
555        台灯
```

如果有重复记录，UNION 会过滤掉重复记录，只保留一条重复记录（**425　mp3**）。

找出两个查询中不相同的记录

查询两个集合中不相同的记录也有 3 种方法。本例将找出表 product_A 和 product_B 中不相同的记录。

■ 方法 1

```
SQL> SELECT id, name--找出只存在表 product_A 中的记录
  2  FROM product_A
  3  WHERE (id,name) not in(SELECT id,name  FROM product_B)

  4  UNION

  5  SELECT id, name--找出只存在表 product_B 中的记录
  6  FROM product_B
  7  WHERE (id,name) not in(SELECT id,name  FROM product_A);
```

SQL 语句的执行结果如下所示：

```
 ID         NAME
---------- ----------
425        落地音响
444        牛奶杯
```

```
555        台灯
```

■ 方法 2

```
SQL> SELECT * FROM--找出只存在表 product_A 中的记录
  2  (
  3  SELECT id, name
  4  FROM product_A
  5  MINUS
  6  SELECT id, name
  7  FROM product_B
  8  )

  9  UNION

 10  SELECT * FROM--找出只存在表 product_B 中的记录
 11  (
 12  SELECT id, name
 13  FROM product_B
 14  MINUS
 15  SELECT id, name
 16  FROM product_A
 17  );
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425        落地音响
444        牛奶杯
555        台灯
```

■ 方法 3

```
SQL> SELECT id,name FROM
  2  (--得到两个集合所有的记录
  3  SELECT id,name
  4  FROM product_A
  5  UNION ALL
  6  SELECT id,name
  7  FROM product_B
  8  )
  9  GROUP BY id,name
 10  HAVING(COUNT(*)=1); --排出重复的记录
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425        落地音响
```

```
444         牛奶杯
555         台灯
```



注：一定要使用 UNION ALL，不能使用 UNION。

找出两个表（查询）中相同的记录

利用 INTERSECT 可以找出两个集合中相同的记录。找出两个表中相同记录的方法比较多，INTERSECT 只是其中一种，下面，我们将对每种方法进行一一介绍。

■ 利用集合运算 INTERSECT

```
SQL> SELECT id,name FROM product_A
  2  INTERSECT
  3  SELECT id,name FROM product_B;
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425     mp3
```

INTERSECT 用于得到两个集合的交集。

■ 利用 IN

如果两条记录的 id 和 name 分别相同，则两条记录相同。

```
SQL> SELECT id,name FROM product_A
  2  WHERE (id,name)IN(SELECT id,name FROM product_B);
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425      mp3
```

■ 利用 EXISTS

```
SQL> SELECT id,name FROM product_A a
  2   WHERE  EXISTS  (SELECT  *  FROM  product_B  b  WHERE  a.id=b.id  and
a.name=b.name );
```


SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425       mp3
```

找出只存在于一个表，而不存在于另外一个表中的记录

本例中将找出存在于表 product_A 中，而不存在于表 product_B 中的记录。两个记录是否相同，以 id 及 name 为判断依据。

■ 利用集合运算 MINUS

MINUS 用于集合相减，下面将用第一个查询的结果集减去第二个集合的结果集。

```
SQL> SELECT id,name FROM product_A
  2  MINUS
  3  SELECT id,name FROM product_B
  4  /
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425        落地音响
444        牛奶杯
```

MINUS 用于两个集合相减。

■ 利用 NOT IN

```
SQL> SELECT id,name FROM product_A
  2  WHERE (id,name) NOT IN (SELECT id,name FROM product_B);
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425        落地音响
444        牛奶杯
```

■ 利用 NOT EXISTS

```
SQL> SELECT id,name FROM product_A a
  2  WHERE NOT EXISTS (SELECT * FROM product_B b WHERE a.id=b.id and
a.name=b.name );
```

SQL 语句的执行结果如下所示：

```
ID         NAME
---------- ----------
425        落地音响
444        牛奶杯
```



16.5.4 删除“容器”中的数据（DELETE 语句）

DELETE 语句用于删除表中数据。若想删除其他用户的表，你必须具有该表上的 DELETE 对象权限。拥有系统权限 DELETE ANY TABLE 的用户可以从任何用户的表中删除数据（数据字典表不能被任何用户删除）。DELETE 语句的简易语法如下：

```
DELETE FROM table_name
WHERE
```

其中，table_name 表示将从该表中删除数据；WHERE 表示删除指定的数据行，如果没有 WHERE 子句，则将删除该表中所有的数据行。

示例 1：删除表中所有的数据行

```
DELETE FROM t_null;
```

示例 2：删除表中特定的数据行



列出要被删除的数据。

```
SQL> SELECT * FROM t_null
  2  WHERE AGE=18;
```

SQL 语句的执行结果如下所示：

```
NAME            AGE
---------- ----------
小布丁             18
小不点             18
小不点             18
```



执行删除。

```
SQL> DELETE FROM t_null WHERE AGE=18;
 3 rows deleted
```



如果想让删除永久化，需要进行提交。

```
SQL> COMMIT;
 Commit complete
```



查看数据删除是否成功。

```
SQL> SELECT * FROM t_null
  2  WHERE AGE=18;
    未选定行
```

1．用最快的方法删除数据（TRUNCATE 语句）

删除表中的数据通常使用 DELETE 命令，DELETE 可以删除表中的部分数据，也可以删除表中的全部数据。DELETE 命令删除表中全部数据的效率较低。接下来，我们将介绍其他快速删除表中**全部数据**的方法。

■ 方法 1：利用 TRUNCATE 命令

```
SQL> TRUNCATE TABLE student2;
Table Truncated
```

TRUNCATE 有如下优点：

- TRUNCATE 会进行空间回收，降低高水位线；
- TRUNCATE 是 DDL 语句；
- TRUNCATE 不会产生回滚信息；
- TRUNCATE 是删除整个表数据最快的一种方法；
- TRUNCATE 对表上的约束（Constraint）、触发器（Trigger）、权限等不会有太大的

影响。

■ 方法 2：重建表

本例将模拟如何快速删除表 student2 中的数据。

STEP 01 创建一个临时表，这个表的结构和要被删除的表完全相同，只是该表中没有任何数据。

```
SQL> CREATE TABLE tem_student2
  2  AS
  3  SELECT * FROM student2
  4  WHERE 1=0;
Table created
```

创建临时表 tem_student2，这个表的结构和 student2 完全相同，只是这个表中**没有数据**。1=0 使创建的临时表 tem_student2 中没有数据。

STEP 02 删除原来的表。

```
SQL> DROP table student2;
Table dropped
```

表 student2 被删除，表占用的空间被释放。

STEP 03 重建原来的表。

```
SQL> CREATE TABLE student2
  2  AS
  3  SELECT * FROM tem_student2;
Table created
```

根据临时表 tem_student2 重建原来的表 student2。

STEP 04 重建表 student2 上的索引、约束和触发器等。

重建表有下面的特点：

- 重建表会进行空间回收，降低高水位线；
- 重建表使用的都是 DDL 语句；
- 重建表不会产生回滚信息；
- 重建表也是删除整个表中数据比较快的一种方法；
- 表重建以后需要重新维护约束、触发器和权限等，这是表重建的缺点。

2．利用 ROWID 和 MAX 函数快速删除重复记录

在 Oracle 中，每一条记录都有一个 ROWID，ROWID 在整个数据库中是唯一的，ROWID 由 Oracle 自己产生和维护，并唯一标识一行（无论该表是否有主键和唯一性约束）。ROWID 确定了每条记录在哪个数据文件、哪个块上。即使两条记录完全相同，它们的 ROWID 也不会相同，所以利用 ROWID 和 MAX（或者 MIN）函数，就可以删除表中的重复记录。

删除重复记录有几种方法。本节将通过删除表 student 中的重复记录为例，并假设这

些记录关于列 ID 重复。

■ 方法 1：利用 NOT IN

```
DELETE FROM student
WHERE ROWID NOT IN(SELECT MIN(ROWID) FROM student GROUP BY ID)
```

■ 方法 2：利用不等于

```
DELETE  FROM  student a
WHERE ROWID !=(SELECT  max(ROWID)
FROM  student b WHERE a.id=b.id)
```

■ 方法 3：利用临时表

STEP 01 创建一临时表 tem_student，该表中存放的数据和表 student 的相同，只是该表不存放重复记录。

```
CREATE TABLE tem_student
AS
(SELECT distinct id, first_name, last_name, major, current_credits
FROM student );
```

STEP 02 删除表 student 中的数据。

```
TRUNCATE table student;
```

这里一定要用 TRUNCATE，TRUNCATE 的速度比较快。

STEP 03 把临时表中的数据插入到原来的表 student 中。

```
INSERT INTO student
SELECT * FROM tem_student;
```



16.6 数据库的核心——约束管理

完整性约束（Integrity Constraints）是一些规则，利用这些规则，对表中一列或者多列的数据进行限制，防止一些非法的数据进入数据库。当然，完整性约束是商业规则的体现。Oracle 中，有 5 种类型的约束。

- 不为空约束（NOT NULL Integrity Constraints）
- 唯一值约束（UNIQUE Key Integrity Constraints）
- 主键约束（PRIMARY KEY Integrity Constraints）
- 参照完整性约束（Referential Integrity Constraints）
- 检查约束（CHECK Integrity Constraints）

可以在创建表时定义约束，也可以对已经存在的表添加约束。



16.6.1 五大约束

接下来，我将对上面提到的约束进行一一介绍。

1．不为空约束（NOT NULL Integrity Constraints）

不为空约束，即当往一个表中插入数据（或者更新数据）时，如果遇到某列上有“不为空约束”存在，则该列不能设置成空，换句话说，就是该列必须有值存在。

2．唯一键约束（UNIQUE Key Integrity Constraints）

唯一键约束要求，在一列（或者多列）中的所有值必须唯一，即没有重复值存在。唯一性约束可以由一列或者多列构成。

3．主键约束（PRIMARY KEY Integrity Constraints）

主键约束由一列或多列构成，其值用来唯一标识表中的一行。创建主键时，将自动创建主键索引，主键索引是唯一索引的特殊类型。主键索引要求主键中的每个值是唯一的。当在查询中使用主键进行查询时，主键索引可以帮助我们快速访问数据。



知识点索引——主键约束和唯一键约束的区别

- ✓ 主键会隐含创建唯一性索引。
- ✓ 主键可作外键，唯一性约束不可以 。
- ✓ 组成主键的任何一列不允许为空（NULL），唯一性约束可以。
- ✓ 每个表上只能有一个主键，却允许有多个唯一性约束。

4．参照完整性约束（Referential Integrity Constraints）

参照完整性约束，是对两个表之间的一种限制（说白了就是外键）。其原理是：如果列 A 是表 R1 的外键，它与表 R2 的主键 K 相对应，则表 R1 中列 A 的每个值要么取空值（NULL），要么等于表 R2 中主键 K 的值。我们把 R2 叫父表，R1 叫子表。

5．检查约束（CHECK Integrity Constraints）

约束检查指定一系列的判定条件，当进行插入或者更新操作时，如果判断条件为真（TRUE），则允许数据的插入和更新。



16.6.2 约束管理

在阅读本节的约束管理以前，请先创建测试表 t_integrity，接下来的例子都是基于该表。

```
CREATE TABLE t_integrity
(id varchar2(10),name varchar2(10),keynum number);
```

1．创建约束

创建主键

有两种方法创建主键，主键也是一种约束，每个约束都有一个名字，本节的两个例子

都会被 Oracle 隐含赋予约束名字。

- 创建表时指定主键

```
CREATE TABLE t_integrity
(id varchar2(10) PRIMARY KEY,name varchar2(10),keynum number);
```

- 在已经存在的表上添加主键

```
ALTER TABLE t_integrity
  ADD PRIMARY KEY (id,name);
```

主键可以由单列组成，亦可由多列构成。

下面的 SQL 语句也可添加主键约束，并指定约束名。

```
ALTER TABLE t_integrity
ADD CONSTRAINT pk3 PRIMARY KEY(id,name);
```

创建唯一性约束

- 创建表时指定唯一性约束

```
CREATE TABLE t_integrity
(id varchar2(10) UNIQUE,name varchar2(10),keynum number);
```

- 在已经存在的表上添加唯一性约束

```
ALTER TABLE t_integrity
ADD UNIQUE (id,name);
```

唯一性约束可以由一列或者多列构成。

下面的 SQL 语句也可添加唯一性约束，并指定约束名。

```
ALTER TABLE t_integrity
ADD CONSTRAINT uk_dept UNIQUE(id,name)
```

其中，uk_dept 是约束的名字。

创建不为空（NOT NULL）约束

- 创建表时指定不为空约束

```
CREATE TABLE t_integrity
(id varchar2(10) NOT NULL,name varchar2(10),keynum number);
```

- 在已经存在的表上添加不为空约束

```
ALTER TABLE t_integrity
MODIFY (name NOT NULL);
```

创建检查约束（CHECK）

- 创建表时指定检查约束

```
CREATE TABLE t_integrity
(id varchar2(10) ,name varchar2(10),
keynum NUMBER CONSTRAINT c_check CHECK(keynum BETWEEN 10 AND 20));
```

c_check 是检查约束的名字，检查约束的规则是：列 keynum 的值必须介于 10 到 20 之间。

➢ 在已经存在的表上添加检查约束

```
ALTER TABLE t_integrity
ADD CONSTRAINT c_check CHECK(keynum BETWEEN 10 AND 20);
```

➢ 往存在检查约束的表中插入数据

```
INSERT INTO t_integrity(id,name,keynum)
VALUES('224','王晓',5)
ORA-02290: 违反检查约束条件 (ITME.C_CHECK)
```

因为插入值 keynum=5 不在指定的范围内（10 到 20），也就违反检查约束，因此，数据插入被拒绝。

2．删除约束

🕮 **删除主键**

➢ 删除表 t_integrity 上的主键

```
ALTER TABLE t_integrity
DROP PRIMARY KEY;
```

➢ 如果表上的主键被其他表所引用（参照），可以加上选项 CASCADE，选项 CASCADE 表示主键和参照完整性约束一同被删除。

```
ALTER TABLE t_integrity
DROP PRIMARY KEY CASCADE;
```

➢ 如果知道主键约束的名字，可以使用下面的 SQL 语句删除主键约束：

```
ALTER TABLE t_integrity
  DROP CONSTRAINT pk3 CASCADE;
```

其中，pk3 是约束的名字。

🕮 **删除唯一性约束**

➢ 删除表 t_integrity 上的唯一键约束。

```
ALTER TABLE t_integrity
DROP UNIQUE (name);
```

➢ 如果知道唯一性约束的名字，可以使用下面的 SQL 语句删除唯一性约束。

```
ALTER TABLE t_integrity
  DROP CONSTRAINT uk_dept CASCADE;
```

本例中，uk_dept 是唯一键约束的名字。

🕮 **删除不为空（NOT NULL）约束**

删除不为空约束（NOT NULL），使其变为空（NULL）。

```
ALTER TABLE t_integrity
   MODIFY (name NULL);
```

🕮 **删除检查约束（CHECK）**

```
ALTER TABLE t_integrity
DROP CONSTRAINT c_check;
```

其中，c_check 是约束的名字。

3．外键的使用（参照完整性）

本节将通过例子讲述参照完整性（外键），本节的例子侧重于练习外键的使用，包括外键对数据插入、删除的影响。

有两个表：学生信息表（子表）和学院信息表（父表），如图 16-5 所示。

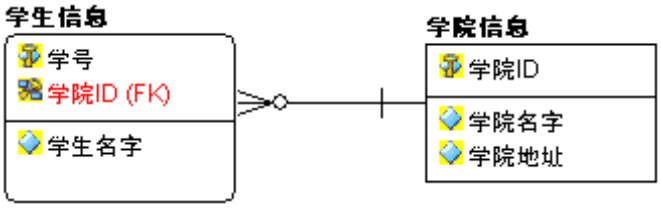


图 16-5　外键关系

“学号”是“学生信息”表的主键，“学院信息”表的主键是“学院 ID”，“学生信息”表的外键是“学院 ID”，其指向“学院信息”表的“学院 ID”。

外键使用的过程说明如下。

STEP 01 创建“学院信息”表（department）——父表。

```
CREATE TABLE department(
   DEPT_ID    CHAR(10)    NOT NULL,
   NAME       CHAR(10),
   ADDRESS    CHAR(10),
   CONSTRAINT PK2 PRIMARY KEY (DEPT_ID)
);
```

学院信息表上添加了主键。

STEP 02 创建“学生信息”（student）表——子表。

```
CREATE TABLE student(
   SID        CHAR(10)    NOT NULL,
   DEPT_ID    CHAR(10)    NOT NULL,
   NAME       CHAR(10),
   CONSTRAINT PK1 PRIMARY KEY (SID)
);
```

STEP 03 给学生信息表添加外键。

```
ALTER TABLE student ADD CONSTRAINT Refdepartment3
   FOREIGN KEY (DEPT_ID)
REFERENCES department(DEPT_ID);
```

给学生信息表（student）添加外键，外键指向学院信息表（department）。

知识点索引——级联删除

如果想在删除**父表**中的值时，也让 Oracle 自动删除**子表**中相应的值，则在创建外键时，需要指定选项 ON DELETE CASCADE。例如：

```
ALTER TABLE student ADD CONSTRAINT Refdepartment3
FOREIGN KEY (DEPT_ID)
REFERENCES department(DEPT_ID) ON DELETE CASCADE;
```

STEP 04 往学生信息表（子表）中插入数据。

```
SQL> INSERT INTO student(SID,NAME,DEPT_ID)
  2  VALUES('112','王峰','001');
  ORA-02291: 违反完整约束条件 (ITME.REFDEPARTMENT3) - 未找到父项关键字
```

SQL 语句显示，不能往学生信息表（student）中插入数据，学院编号 001 在学院信息表（department）中不存在。根据外键的功能，在父表（department）中如果相应数据不存在，则不允许往子表（student）中插入数据。

STEP 05 往学院信息表（父表）中插入数据。

```
INSERT INTO department(DEPT_ID,NAME,ADDRESS)
VALUES('001','经济管理学院','东校区');
COMMIT;
```

学院（编号是 001）的信息被插入。

STEP 06 再次往学生信息表（子表）中插入数据。

```
SQL> INSERT INTO student(SID,NAME,DEPT_ID)
  2  VALUES('112','王峰','001');
 1 row inserted
 SQL> COMMIT;
```

插入成功。再往学生信息表中插入另外一条记录。

```
SQL> INSERT INTO student(SID,NAME,DEPT_ID)
  2  VALUES('233','高老大','001');
1 row inserted
 SQL> COMMIT;
```

STEP 07 查看两个表中的数据。

```
SQL> SELECT * FROM department;
```

SQL 语句的执行结果如下所示：

```
 DEPT_ID            NAME               ADDRESS
-------------------- -------------------- --------------------
001                经济管理学院         东校区
SQL> SELECT * FROM student;
```

SQL 语句的执行结果如下所示：

```
SID                DEPT_ID             NAME
-------------------- -------------------- --------------------
112                001                王峰
233                001                高老大
```

STEP 08 删除（DELETE）子表（student）中的信息。

```
SQL> DELETE FROM student WHERE name='高老大';
 1 row deleted
```

删除子表信息，对主表没有任何影响。

STEP 09 删除（DELETE）父表的信息。

```
SQL> DELETE FROM department WHERE DEPT_ID='001';
ORA-02292: 违反完整约束条件 (ITME.REFDEPARTMENT3) - 已找到子记录
```

不能删除父表（department）信息，因为子表（student）中存在对应的记录（除非在

创建外键时指定了选项 ON DELETE CASCADE，则 Oracle 可以自动实现级联删除）。

STEP 10 删除（DROP）父表。

```
SQL> DROP TABLE department;
ORA-02449: 表中的唯一/主键被外键引用
```

不能删除父表，因为父表上有其他表指向它的外键。若想删除父表，必须解除其他表（子表）对它的引用（指向）。解除子表（student）对父表（department）的引用：

```
ALTER TABLE student DROP CONSTRAINT Refdepartment3;
```

删除约束 Refdepartment3，约束 Refdepartment3 是在前面的步骤中创建的。外键实际上也是一种约束（CONSTRAINT）。

再次删除主表。

```
SQL> DROP TABLE department;
Table dropped
```

删除主表成功。

也可以使用下面的命令一次性删除主表。

```
DROP TABLE department CASCADE CONSTRAINTS;
```

CASCADE CONSTRAINTS 表示删除父表的同时，删除其他子表上相关的参照完整性约束。

子表的删除（DROP）对主表没有任何影响。



16.7 数据库的核心——LOB 管理

大对象数据类型（Large Object Data Types，LOB）用于存储大文本、图像、视频、声音等信息。大对象数据类型专为大容量存储而设计。Oracle 预定义了 4 种大对象类型，它们是 BFILE、BLOB、CLOB、NCLOB。

LOB 定位器（LOB Locators）：LOB 数据是我们存入到数据库中的值，而 LOB 定位器是操作这些值的指针。当我们想读取或者修改 LOB 数据时，必须先要获取 LOB 定位器，然后通过 LOB 定位器才能读取或者修改 LOB 数据。说白了，LOB 定位器就是指向 LOB 数据的指针。

处理 LOB 数据的方法有很多种，通常，通过 DBMS_LOB 包来检索或操作 LOB 数据。DBMS_LOB 包中有 READ()、APPEND()、WRITE()、ERASE()、COPY()、GETLENGTH()、SUBSTR()等函数，可以很方便地操作 LOB 对象。各个函数的含义大致说明如下：

- OPEN 打开 LOB。
- CLOSE 关闭 LOB。
- APPEND 将源 LOB 中的内容追加到目标 LOB 中。
- COPY 从源 LOB 中的一部分或者全部数据复制到目标 LOB。
- ERASE 删除 LOB 中全部或部分内容。
- TRIM 将 LOB 值减少到指定的长度。
- READ 从 LOB 中读取数据。

- WRITE 向 LOB 中写入数据。
- COMPARE 比较两个同种数据类型的 LOB 的部分或全部值是否相同。
- GETLENGTH 获取 LOB 的长度。
- FILEOPEN 打开文件。
- FILECLOSE 关闭文件。
- FILEEXISTS 检查文件的存在性。

LOB 数据通常和普通的数据分开存储，如下面的 SQL 语句所示，LOB 可以与主表分别存放在不同的表空间。

```
CREATE TABLE lob_test
(bf bfile,
bl  blob,
cl  clob,
ncl nclob)
LOB (cl) STORE AS (tablespace users)
LOB (bl) STORE AS (tablespace myspace);
```

其中，bl 存储在表空间 users；cl 存储在表空间 myspace。



16.7.1 BFILE

我们可以把大的二进制对象存储在数据库之外（也就是以操作系统文件的形式进行存储），BFILE 存放在数据库中，BFILE 存放的是文件指针（File Locator），该指针指向数据库之外的操作系统文件，数据实际存放在操作系统中。BFILE 是只读的，不能修改它。DBA 必须保证 Oracle 有权限读取 BFILE 指向的操作系统文件。这些文件的完整性由操作系统来保证。

BFILE 指向的文件存储在数据库之外，因此，BFILE 不参与数据库的事务，也就是不能 ROLLBACK 或 COMMIT，总而言之，ROLLBACK 和 COMMIT 对 BFILE 指向的文件没有影响。



16.7.2 BLOB

BLOB 用于存储非结构化的二进制数据，最长可达 128TB，数据存储在数据库中。BLOB 数据类型通过 LOB 定位器（指定数据地址）存储在表列中，实际的 BLOB 数据存储在独立的表空间中。创建含有 BLOB 列的表时，可以指定一个单独的表空间来存放 BLOB 数据。



16.7.3 CLOB

CLOB 存放数据库字符集（Database Character Set Data）格式的字符数据，最大长度可达 128TB，数据存储在数据库中。CLOB 数据类型通过 LOB 定位器（指定数据地址）

存储在表列中，实际的 CLOB 数据存储在独立的表空间中。创建含有 CLOB 列的表时，也可以指定一个单独的表空间来存放 CLOB 数据。

16.7.4 NCLOB

NCLOB 的功能和 CLOB 的一样，只是 NCLOB 用于存放统一编码国家字符集（Unicode National Character Set）数据。

16.7.5 BFILE 应用实战

本文以学生简历为例。有一学生信息表 BFILE_student，name 表示学生的姓名，resume 是文件指针，指向学生的简历。学生的简历以文件的形式存放在数据库之外，操作系统中（本文假设学生的简历存放在磁盘目录 E:\resume 之中）。BFILE 应用实战的例子说明如下。

STEP 01 创建学生信息表。

```
CREATE TABLE BFILE_student
(  name varchar2(10),
   resume  BFILE
);
```

STEP 02 创建目录对象 DIR_RESUME。

```
CREATE DIRECTORY DIR_RESUME AS 'E:\resume';
```

此命令只是创建了一个数据库对象，并没有创建 E:\resume 目录。用户需要进入到操作系统中创建此目录，并使 Oracle 对该目录有读的权限。本例中的目录已经存在，并且该目录下有一个简历文件 liuhongresume.doc。

STEP 03 使用户获得目录的访问权限。

```
GRANT READ ON DIRECTORY DIR_RESUME TO itme;
```

把目录读的权限授予用户 itme。

STEP 04 往表中插入数据。

```
INSERT INTO BFILE_student(name,resume) VALUES(' 刘 宏 ',BFILENAME('
DIR_RESUME ', liuiaoresume.doc'));
```

BFILENAME 函数用于返回文件指针。本例中，文件 liuiaoresume.doc 实际不存在。

STEP 05 更改指针，使其指向另外一个文件 liuhongresume.doc。

```
UPDATE BFILE_student SET resume=BFILENAME(' DIR_RESUME ', liuhongresume.
doc')
WHERE name='刘宏';
```

STEP 06 BFILE 在 PL/SQL 块中的使用。

本段程序块的目的在于练习在程序开发时，如何在程序中使用 BFILE。

```
DECLARE
--定义变量，数据类型为文件指针
s_bfile   BFILE;
BEGIN
  --获得文件指针，只有获得文件指针才能对文件进行操作
  SELECT resume INTO s_bfile FROM BFILE_student WHERE name='刘宏';
  --通过指针 s_bfile 以只读方式打开文件，
  DBMS_LOB.FILEOPEN(s_bfile,DBMS_LOB.FILE_READONLY);

  --…
  --对文件进行各种操作，如使用 DBMS_LOB.LOADCLOBFROMFILE 读文件的内容
  --…
  --文件使用完成后，关闭文件
  DBMS_LOB.CLOSE(s_bfile);
END;
```



16.7.6 CLOB 应用实战

因为 BLOB、CLOB、NCLOB 在使用上都有许多相似之处，因此，我们以 CLOB 为例，演示它们的用法。在实验之前，我们需要创建学生论文表（student_paper），该表用于存放学生的名字及学生的论文。需要注意的是，列 paper 是 CLOB 类型。

```
CREATE TABLE student_paper (
  student_name VARCHAR2(20),--学生姓名
  paper CLOB--学生的毕业论文
  );
```



插入学生信息（插入 LOB 数据）。

```
INSERT INTO student_paper(student_name, paper)
VALUES ('王晶',EMPTY_CLOB( ));
COMMIT;
```

empty_clob()是 Oracle 的内置函数，它创建了一个空的 LOB 定位器（或者叫 LOB 定位符，LOB locator），它没有指向任何数据，实际上是一个空指针。可以使用 IS NULL 判断 LOB 变量是否为空。这里没有插入数据，只是插入了一个空指针。



填充学生的论文（向 CLOB 中写入数据）。

```
DECLARE
      v_clob CLOB;--定义 CLOB 的指针变量（也就是 LOB 定位器）
BEGIN

      --得到 CLOB 的指针，并存放在变量 v_clob 中。这里要加上 FOR UPDATE 之句，表示一个事务的开始，否者 dbms_lob.OPEN 的执行会产生错误“ORA-22292: Cannot open a LOB in read-write mode without a transaction”
      SELECT paper
```

```
         INTO v_clob
         FROM student_paper
        WHERE student_name = '王晶'
          FOR UPDATE;
       --根据 CLOB 指针以读写方式打开 CLOB
       dbms_lob.OPEN(v_clob, dbms_lob.lob_readwrite);
       --往 LOB 中写入字符 mis，3 表示被写的字符数量，1 表示开始写的位置
       --下面的语句表示字符串中只有 3 个字符被写入 LOB 中
       dbms_lob.WRITE(v_clob, 3, 1, 'mis');
       --往 LOB 的末尾追加字符，20 表示被写的字符数量
      --下面的语句表示字符串中只有 20 个字符被写入 LOB 中
       dbms_lob.writeappend(v_clob,                                    20,
'understandfsdfsdfsdfsdfsdfsdfdsfdssfsdfsdfdsfdsererwerwerew');
       --关闭 LOB 指针
       dbms_lob.CLOSE(v_clob);
       --提交修改
       COMMIT;
     EXCEPTION
       WHEN OTHERS THEN
       --出现错误时，提示失败
         dbms_output.put_line('FAILED');
         ROLLBACK;
 END;
```

STEP 03 从 CLOB 读取数据，并显示。

```
DECLARE
         vclob2 CLOB;
         v_out VARCHAR2(1000); --存放读取的字符
         V_read_lenth NUMBER; --读取的字符长度
BEGIN
         --获得一个 lob locator
         SELECT paper
           INTO vclob2
           FROM student_paper
          WHERE student_name='王晶';
         --将要读取 13 个字符
         V_read_lenth := 13;
         --1 表示从第 1 个字符开始读取，读取的字符放在变量 v_out 中
         DBMS_LOB.READ(vclob2,V_read_lenth,1, v_out);
        --显示读取的结果
        DBMS_OUTPUT.PUT_LINE(v_out);
        DBMS_OUTPUT.PUT_LINE(length(v_out));
END;
```

执行结果如下所示：

```
misunderstand
```

```
13
PL/SQL procedure successfully completed
```



16.8 用户自定义数据类型（User-defined Data Type）

除了标准的数据类型，用户还可以定义自己的数据类型，我们把这样的数据类型叫做“用户自定义数据类型（User-defined Type）”。Oracle 支持对象类型（Object Type）、嵌套表类型（Nested Table Data Type）和可变数组类型（Varray Data Type）三种用户自定义数据类型。



16.8.1 对象类型（Object Type）

对象类型用于描述客观世界中存在的事物，例如客户、订单。对象类型是在其他内置数据类型基础上建立的（当然，也可以根据一种对象类型建立另外一种对象数据类型）。对象类型形象地刻画了真实世界中的物体。

对象类型中可以包含多个元素，我们称之为“属性”，这些属性共同构成了结构化的数据单元，例如订单编号就是订单的属性之一。

对象类型中还可以包含多种操作，我们把这些操作称为**方法**，用户可以对一个数据单元执行某种方法。

这里描述的对象类型和其他语言描述（C++）对象很相似，一个对象有属性，可以使用对象的方法，对对象进行操作。实际上，这里的“方法”其实就是一些存储过程、函数。

创建对象类型

本书假设学校是一个对象类型。

```
CREATE OR REPLACE TYPE schoolTYPE AS OBJECT
(
name varchar2(30),
City varchar2(30),
Street varchar2(30)
);
```

创建对象类型 schoolTYPE，该对象类型包括 3 个属性：

- name 表示学校的名字；
- City 表示学校所在的城市；
- Street 表示学校所在的街道。

删除对象类型

```
DROP   TYPE schoolTYPE;
```

如果一个对象类型已经被其他表或者类型使用，则不能被删除。

查看对象类型的结构

可以使用 DESC 命令查看对象类型的结构信息。

```
SQL> DESC  schoolTYPE
```

SQL 语句的执行结果如下所示：

```
名称                                      是否为空? 类型
 ----------------------------------------- --------
 NAME                                               VARCHAR2(30)
 CITY                                               VARCHAR2(30)
 STREET                                             VARCHAR2(30)
```

对象的存储形式

对象有以下两种存储形式：

- 对象单独存储在关系表中，也就是整个表只存储对象，这样的表称为**对象表**（Object Table）。在这样的表中，每行表示一个对象。存储于对象表中的对象被称为行对象（Row Object）。
- 与表中其他数据一起存储，即，对象只占表中的一列。我们把这样的对象称为列对象（Column Object）。

基于对象类型创建表

示例 1：创建对象表——学校信息表 school_infor

```
CREATE TABLE school_infor of schoolTYPE;
```

示例 2：创建含有对象的关系表——学生信息表 student_infor

```
CREATE TABLE student_infor
(name varchar(10),
id number,
school schoolTYPE);
```


往表中插入数据

- 往表 school_infor 中插入数据

```
INSERT INTO school_infor VALUES (
schoolTYPE('北大', '北京', '中关村'));
commit ;
```

- 往表 student_infor 中插入数据

```
INSERT INTO student_infor VALUES ('小明',11111,schoolTYPE('北大', '北京', '
中关村'));
commit ;
```

这里要特别注意的是，**schoolTYPE** 是一种数据类型。

查询与对象相关的表

➢ 查询学校信息

```
SQL> SELECT * FROM school_infor;
```

SQL 语句的执行结果如下所示：

```
NAME                          CITY         STREET
   ----------------------------------------------------------------
北大                          北京         中关村
```

➢ 查询学校信息（指定字段名）

```
SELECT name,city, street  FROM school_infor;
```

SQL 语句的执行结果如下所示：

```
NAME                          CITY         STREET
-------------------------------------------
北大                          北京         中关村
```

➢ 查询学生信息表

```
SQL> SELECT * FROM student_infor;
```

SQL 语句的执行结果如下所示：

```
NAME             ID     SCHOOL(NAME, CITY, STREET)
-----------------------------------------------------------
小明           11111     SCHOOLTYPE('北大', '北京', '中关村')
```

➢ 查询学生信息表（指定字段名）

```
SELECT stu.name,stu.id,stu.school.name,stu.school.city,stu.school.street
FROM student_infor stu;
```

SQL 语句的执行结果如下所示：

```
NAME    ID    SCHOOL.NAME      SCHOOL.CITY           SCHOOL.STREET
------------------------------ ------------------------------
小明   11111         北大          北京                          中关村
```


16.8.2 可变数组类型（Variable-size Arrays Type）

可变数组是一个有序的相同数据类型元素的集合。一个数组内所有数据元素的数据类型均相同。每个元素均有一个索引编号（下标），代表元素在数组中的位置。通过下标访问数组元素。值得一提的是，此处所谓的“索引”不是我们通常所说的索引。定义可变数组时，需要指定元素的最大数量。

可变数组数据通常采用内联存储（Stored In Line），即与其在同一表中的其他数据存储

在同一表空间内。但是，如果可变数组数据很大，也可以将其存储在 BLOB 中。

创建可变数组类型

➢ 创建可变数组类型——使用基本数据类型

```
CREATE TYPE address_list AS VARRAY(10) OF VARCHAR2(80);
```

本例中，最大元素数量是 10，每个元素的数据类型都是 VARCHAR2(80)。

这里并没有创建表，只是创建一种新的数据类型！

➢ 创建可变数组类型——使用对象类型



创建对象类型——电话信息。

```
CREATE TYPE phone_typ AS OBJECT (
country_code VARCHAR2(2),
area_code VARCHAR2(3),
phone_number VARCHAR2(7));
```

其中，country_code 表示国家代码；area_code 表示区号；phone_number 表示电话号码。



创建可变数据类型。

```
CREATE TYPE varray_phone AS VARRAY(10) OF phone_typ;
```

创建可变数组类型 varray_phone，varray_phone 有 10 个元素，每个元素的数据类型是 phone_typ，phone_typ 是对象类型。

这里并没有创建表，只是创建一种新的数据类型！

删除可变数组类型

```
DROP  TYPE address_list;
```

如果一个可变数组类型已经被其他表或者类型使用，则不能被删除。

创建含有可变数组类型的表

假设在大学里，一个导师可能有多个电话号码。接下来，我们将创建导师表 instructor，其中 name 为导师姓名，phone_list 为导师电话信息列表。

```
CREATE TABLE instructor(
name varchar2(10),
phone_list varray_phone);
```

往含有可变数组类型的表中插入数据

```
INSERT INTO instructor VALUES (
'王教授',
varray_phone(
phone_typ ('86', '010', '6634553'),
phone_typ ('86', '020', '3792345'),
phone_typ ('86', '022', '5632199')));
```

```
COMMIT;
```

往表 instructor 中插入一条数据，本例中，“王教授”有 3 个电话号码。继续插入另外一条记录：

```
INSERT INTO instructor VALUES (
'张教授',
varray_phone(
phone_typ ('86', '010', '4634553'),
phone_typ ('86', '010', '4792345'),
phone_typ ('86', '010', '4632199')));

COMMIT;
```

查询含有可变数组类型的表

- 列出所有导师的信息

```
SQL> SELECT * FROM instructor;
```

SQL 语句的执行结果如下所示：

```
NAME   PHONE_LIST(COUNTRY_CODE, AREA_CODE, PHONE_NUMBER)
--------------------------------------------------------------------------
王教授
VARRAY_PHONE(PHONE_TYP('86', '010', '6634553'), PHONE_TYP('86', '020',
'3792345'
), PHONE_TYP('86', '022', '5632199'))
张教授
VARRAY_PHONE(PHONE_TYP('86', '010', '4634553'), PHONE_TYP('86', '010',
'4792345'
), PHONE_TYP('86', '010', '4632199'))
```

- 列出张教授的所有电话信息

```
SELECT phone_list FROM instructor
WHERE name='张教授';
```

SQL 语句的执行结果如下所示：

```
PHONE_LIST(COUNTRY_CODE, AREA_CODE, PHONE_NUMBER)
-----------------------------------------------------------------
VARRAY_PHONE(PHONE_TYP('86', '010', '4634553'), PHONE_TYP('86', '010',
'4792345'
), PHONE_TYP('86', '010', '4632199'))
```

- 列出王教授区号是 020 的所有电话信息

```
SELECT country_code,area_code, phone_number
FROM TABLE(SELECT phone_list FROM instructor
```

```
    WHERE name='王教授')
    WHERE area_code='020';
```

SQL 语句的执行结果如下所示：

```
CO ARE PHONE_N
-- --- -------
86 020 3792345
```


注意：本例中的“TABLE”是关键字，TABLE 允许在 FROM 子句中查询集合。

得到所有老师所有的电话号码

```
    SELECT e.*
    FROM instructor d, TABLE(d.phone_list) e;
```

SQL 语句的执行结果如下所示：

```
CO ARE PHONE_N
-- --- -------
86 010 6634553
86 020 3792345
86 022 5632199
86 010 4634553
86 010 4792345
86 010 4632199
```

更新含有可变数组类型的表

王教授搬到另外一个城市，需要变更他家的区号，并且停掉一部电话。

```
    UPDATE instructor
    SET phone_list=varray_phone(phone_typ('86', '023', '4634553'),phone_typ
('86', '023', '4634553'))
    WHERE name='王教授'
```

特别要注意的是 varray_phone 及 phone_typ 都是用户定义的数据类型。varray_phone 是可变数组类型，phone_typ 是对象类型。同时不能更新 VARRAY 中的单个元素，必须更新整个 VARRAY。

16.8.3 嵌套表类型（Nested Table Type）

嵌套表（Nested Table）是一个**无序**的相同类型数据元素的集合。需要注意的是，所有的元素具有相同的数据类型。定义嵌套表时，无须指定元素的最大数量。元素的数据类型可以是标准的 Oracle 数据类型，也可以是用户自定义的数据类型（如对象类型）。使用 CREATE TYPE ... AS TABLE OF 定义嵌套表。CREATE TYPE ... AS TABLE OF 只是定义一种新的数据类型，并未涉及任何空间的分配。

创建含有嵌套表类型的表

我们都知道，一个学生可以有多本书，在此，我们把书作为一个对象类型（bookOBJTYPE），把一个学生拥有的书籍作为一个嵌套表类型（bookNESTEDTYPE）,然后创建学生图书表（studentbook）。


创建对象类型 bookOBJTYPE。

```
CREATE OR REPLACE TYPE bookOBJTYPE AS OBJECT
(
bookid varchar2(10),
bookname varchar2(18),
describe varchar2(10))
not final;
```


创建嵌套表类型 bookNESTEDTYPE。

```
CREATE OR REPLACE TYPE bookNESTEDTYPE AS TABLE OF bookOBJTYPE;
```

上述命令只是创建了一类新的数据类型，并未创建表。


创建学生图书表 studentbook。

```
CREATE TABLE studentbook
(
student_name varchar2(9) Not Null Primary Key,
home_address varchar2(9),
booklist bookNESTEDTYPE
)TABLESPACE jspace
Nested Table booklist STORE AS booklist_table (TABLESPACE myspace);
```

其中，student_name 表示学生姓名；home_address 表示学生家庭地址；Booklist 表示学生拥有的书籍，数据类型是 studentbook；jspace 表示存放主表数据（studentbook）的表空间。嵌套表的主表和子表分开存储，子表存放在主表之外；myspace 表示存放子表数据（booklist_table）的表空间。

嵌套表的存放空间

■ 查看主表的存放空间

```
SQL> SELECT tablespace_name
  2    FROM dba_tables
  3    WHERE table_name = 'STUDENTBOOK';
```

SQL 语句的执行结果如下所示：

```
TABLESPACE_NAME
------------------------------
JSPACE
```

可见主表存放在表空间 JSPACE 中。

■ 查看子表的存放空间

```
SQL> SELECT tablespace_name
  2   FROM Dba_Segments
  3  WHERE segment_type = 'NESTED TABLE'
  4    AND segment_name = 'BOOKLIST_TABLE';
```

SQL 语句的执行结果如下所示：

```
TABLESPACE_NAME
------------------------------
MYSPACE
```

子表存放在表空间 MYSPACE 中。

删除嵌套表类型

```
DROP   TYPE bookNESTEDTYPE;
```

如果一个嵌套表类型已经被其他表或者类型使用，则不能被删除。

插入数据

```
INSERT INTO studentbook VALUES (
'王小小',
'上海',
bookNESTEDTYPE
(
bookOBJTYPE ('111', '《中国通史》', '历史类'),
bookOBJTYPE ('333', '《火星探秘》', '科普类')
)
);
INSERT INTO studentbook VALUES (
'刘晶',
'天津',
bookNESTEDTYPE
(
bookOBJTYPE ('411', '《网络基础》', '计算机类'),
bookOBJTYPE ('422', '《开发基础》', '计算机类'),
bookOBJTYPE ('433', '《小学语文》', '教材类')
)
);
COMMIT;
```

以上 SQL 语句往表 studentbook 中插入两条数据。学生王小小拥有 2 本书，刘晶拥有 3 本书。

BookNESTEDTYPE 及 bookOBJTYPE 都是用户定义数据类型。bookNESTEDTYPE 是嵌套表类型；bookOBJTYPE 是对象类型。

查询嵌套表

STEP 01 查询所有学生拥有的书籍。

```
SELECT * FROM studentbook
```

STEP 02 查询学生刘晶拥有的书籍。

```
SELECT * FROM TABLE(SELECT T.booklist  FROM studentbook t
WHERE student_name ='刘晶')
```

需要注意的是，TABLE 是关键字。

更新嵌套表

刘晶同学把自己的书籍《小学语文》换成了《大学语文》，其他书籍不变。

```
    UPDATE Table(SELECT t.booklist FROM studentbook t  WHERE t.student_name='
刘晶') tt
    SET tt.bookname='《大学语文》'
    WHERE tt.bookname='《小学语文》';
    COMMIT;
```

删除嵌套表中数据

刘晶同学丢失了《开发基础》这本书。

```
    DELETE
    FROM Table(SELECT t.booklist FROM studentbook t  WHERE t.student_name='
刘晶') tt
    WHERE tt.bookname='《开发基础》';
    COMMIT;
```



16.9 分布式应用

分布式应用是基于分布于全球的 Oracle 数据库（或者非 Oracle 数据库）的应用，这些数据库系统通过网络连接在一起。我们把组成分布式应用的一台数据库系统叫一个站点。一个站点可能分布在一个国家（地区，城市）。每个站点就是一个独立的数据库系统，它们拥有各自的数据库、中央处理机、物理磁盘等。分布式数据库系统在逻辑上属于同一系统，但在物理结构上分布、独立。分布式数据库系统是数据库技术与网络技术结合的产物。它分成两类：一类在物理上分布，但逻辑上却集中；另一类在物理上和逻辑上都分布。

数据库链接（Database Link）是为了访问远程数据库而创建的数据通信链路。数据库链接隐藏了对远程数据库访问的复杂性。通常，我们把正在登录的数据库称为本地数据库，把另外的数据库称为远程数据库。通过数据库链接，可以从本地数据库访问远程数据库中的表，以实现分布式应用，这正是分布式应用的体现。



16.9.1 数据库链接分类

数据库链接分成 3 类：私有数据库链接（Private Database Link）、公有数据库链接（Public Database Link）、全局数据库链接（Global Database Link）。

- 私有数据库链接：只有创建该数据库链接的用户才可以使用它。
- 公有数据库链接：该数据库的所有用户均可以使用它。
- 全局数据库链接：当 Oracle 网络使用目录服务器（Directory Server）时，目录服务器自动创建全局数据库链接，并且统一控制。因此，任何数据库中的用户都可以使用全局

数据库链接访问远程数据库。

16.9.2 创建数据库链接

STEP 01 登录本地数据库。

STEP 02 创建私有数据库链接。

```
CREATE DATABASE LINK remotedb
CONNECT TO itme IDENTIFIED BY zero
USING 'MYZDB';
```

STEP 03 创建公有数据库链接。

```
CREATE PUBLIC DATABASE LINK  remotedb2
CONNECT TO itme IDENTIFIED BY zero
USING 'MYZDB'
```

其中，remotedb2 是数据库链接的名字；itme/ zeros 是远程数据库的用户名/密码；MYZDB 是网络服务名，指定连接到哪个远程数据库（本例中假设 MYZDB 指向远程数据库）。指定的网络服务名和 SQL*Plus 登录时使用的网络服务名是相同的。所以，创建数据库链接以前，必须先配置好网络服务名。有关本例中的网络服务名（MYZDB）配置，**请参考本书 4.2.1 节的内容。**

16.9.3 数据库链接的应用

使用 DML 语句时，通过@符号引用远程数据库。Oracle 允许对远程数据库执行 DML 操作，然而，不允许对远程数据库执行 DDL 操作。执行下面语句以前，需要登录本地数据库。

STEP 01 往远程数据库的表 kk 中插入数据。

```
INSERT INTO kk@remotedb2
VALUES('sssssss');
```

STEP 02 从远程数据库的表 kk 中检索数据。

```
SELECT * FROM kk@remotedb2;
```

STEP 03 更改远程数据库的表 kk 中数据。

```
UPDATE kk@remotedb2 SET a='i am a girl!';
```

STEP 04 删除远程数据库的 kk 表中的数据

```
DELETE FROM kk@remotedb2;
```

16.9.4 删除数据库链接

STEP 01 删除私有数据库链接。

```
DROP DATABASE LINK remotedb;
```

STEP 02 删除公有数据库链接。

```
DROP PUBLIC DATABASE LINK remotedb2;
```



16.10 分区表管理

当今的时代，全球化冲击着世界的每一个角落，许多公司的业务已经遍及全球，超大型数据库（Very Large Database，VLDB）的出现亦不足为奇，一个表的大小常常以 MB、GB 为度量单位，甚至以 TB 为计量单位。庞大的表导致数据库的性能下降，同时，表的可维护性也变得越来越差。为了解决这些问题，Oracle 公司推出了一项强大的技术——分区（Partition）。

使用分区技术，Oracle 允许把一个大表分成几部分，每部分叫一个分区，然后把每个部分放在不同的物理磁盘，以提高整个数据库的性能。每个分区还可以再分成几份，这样产生的分区叫子分区（Subpartition）。无论你怎么进行划分，分区表逻辑上还是一个整体。

分区技术有下面的优点。

- 分区技术使数据库的可管理性变得更加容易，如：用户可以往一个单独的分区中装载数据，而对其他分区没有任何影响；用户可以在单独的分区上创建索引等。
- 分区可以提高表的查询性能，SQL 语句的 WHERE 子句会过滤掉不需要的分区，Oracle 不会再扫描那些不需要的分区。
- 分区技术减少数据的不可用时间，用户可以单独维护一个分区中的数据，而不影响其他分区中数据的使用。
- 分区技术在数据库级完成，几乎不需要对应用程序做任何修改。

根据不同的需求，Oracle 提供下面几种分区方法。

- 范围分区（Range Partitioning）：根据表中列值的范围将整个表分成不同的部分，如按照时间进行范围分区。
- 列表分区（List Partitioning）：使用列表值将表划分成几部分。
- 哈希分区（Hash Partitioning）：使用哈希函数把表分成几部分。
- 复合分区（Composite Partitioning）：同时使用两种分区方法对表进行分区。

1．创建分区表（Partitioned Table）

本节将介绍如何创建各种类型的分区表。

创建范围划分（Range Partition）分区表

示例：创建范围划分分区表 people

```
CREATE TABLE people
( id NUMBER,
age INT NOT NULL,
address VARCHAR2(100))
```

```
PARTITION BY RANGE (age)
(PARTITION P1 VALUES LESS THAN (10)
TABLESPACE users ,
PARTITION P2 VALUES LESS THAN (20)
TABLESPACE myspace,
PARTITION P3 VALUES LESS THAN (30)
TABLESPACE users,
PARTITION P4 VALUES LESS THAN (70)
TABLESPACE myspace);
```

其中，PARTITION BY 指定分区的方式；RANGE 表示分区的方式是范围划分，本例按照年龄（age）进行范围划分；PARTITION 用于指定分区的名字，第一个分区的名字是 P1，第二个分区的名字是 P2；VALUES LESS THAN 指定分区的上界（上限）。第一个分区的上限是 10；第二个分区的上限是 20；最后一个分区的上限是 70。对于最后一个分区的上限，用户也可以使用关键字 MAXVALUE 表示最大值；TABLESPACE 用于指定每个分区存放的表空间。出于性能考虑，每个分区已经放在单独的表空间中。

创建哈希划分（Hash Partition）分区表

哈希划分有两种方法，按照分区数量和指定分区。

- 按照分区数量

```
CREATE TABLE people2 (id NUMBER,age  NUMBER)
PARTITION BY HASH(age) PARTITIONS 4;
```

本例中共有 4 个分区。

- 指定分区

```
CREATE TABLE people3 (id NUMBER,age  NUMBER)
PARTITION BY HASH(age)
(PARTITION pt1 TABLESPACE users,
PARTITION pt2 TABLESPACE myspace,
PARTITION pt3 TABLESPACE users,
PARTITION pt4 TABLESPACE myspace);
```

其中，PARTITION BY HASH 表示按照哈希（Hash）进行划分；PARTITION 指定分区的名字：TABLESPACE 指定每个分区存放的表空间。

创建列表划分（List Partition）分区表

```
CREATE TABLE people4(name VARCHAR2(20),city VARCHAR2(20))
 PARTITION BY LIST (city)
 (
 PARTITION p1
 VALUES ('吉林','大连')
 TABLESPACE users,
 PARTITION p2
 VALUES ('成都','贵州')
 TABLESPACE myspace,
```

```
 PARTITION p3
 VALUES ('广州','桂林', '台北')
 TABLESPACE myspace
 );
```

其中，PARTITION BY LIST 表示按列表进行划分；PARTITION 指定分区的名字；TABLESPACE 指定分区存放的表空间。

2．往分区表中插入数据

往分区表中插入数据与往普通表中插入数据没有什么区别，插入数据时，Oracle 会根据分区的定义自动把数据插入到指定的分区中。

例：

```
INSERT INTO people4 (name ,city )
 VALUES('许晶','成都');
```

3．查询指定分区中的数据

从分区表（people4）指定的分区 p2 中检索数据，PARTITION 是关键字，指定分区的名字。

```
SQL> SELECT * FROM people4 PARTITION(p2);
```

SQL 语句的执行结果如下所示：

```
NAME                 CITY
-------------------- --------------------
许晶                 成都
```

4．修改指定分区中的数据

修改分区表（people4）指定的分区 p2 中的数据。

```
UPDATE  people4 PARTITION(p2) SET name='许晓晶' WHERE name='许晶';
```

5．删除指定分区中的数据

从分区表（people4）指定的分区 p2 中删除数据。

```
DELETE FROM people4 PARTITION(p2) WHERE name='许晶';
```

6．添加（ADD）分区

- 往范围划分分区表 people 中增加新的分区 pa，90 是范围值（上限值），分区 pa 放在表空间 users 中。

```
ALTER TABLE people ADD PARTITION pa VALUES less than(90)  TABLESPACE users;
```

- 往哈希(hash)分区表 people2 中添加新的分区 P_hash，P_hash 存放在表空间 myspace 中。

```
ALTER TABLE people2 ADD PARTITION P_hash TABLESPACE myspace;
```

➢ 往列表划分分区表 people4 中添加新的分区 P_list，P_list 存放在表空间 myspace 中。

```
ALTER TABLE people4 ADD PARTITION P_list VALUES('西宁','银川') TABLESPACE
myspace;
```

7．截断（TRUNCATE）分区

截断分区 p1 中的数据（会释放空间）。

```
ALTER TABLE people TRUNCATE PARTITION p1;
```

也可以使用下面的命令删除分区 p1 中的数据（不会释放空间）：

```
DELETE FROM people PARTITION(p1);
```

8．合并（MERGE）分区

合并范围划分的两个分区，P2 是新的分区名字。

```
ALTER TABLE people MERGE PARTITIONS P1,P2 INTO PARTITION P2;
```

合并列表划分的两个分区（P1、P2），P2 是新的分区名字

```
ALTER TABLE people4 MERGE PARTITIONS p1 ,p2  INTO PARTITION p2;
```



注意：不能合并按照 Hash 进行划分的两个分区。

9．拆分（SPLIT）分区

拆分分区即把一个分区拆分成两个分区。

➢ 拆分按照范围进行划分的分区表 people，将分区 P2 以值 5 为中间线，拆分成两个分区 P1 和 P2。

```
ALTER TABLE people SPLIT PARTITION P2 AT (5) INTO (PARTITION P1,PARTITION
P2);
```

➢ 拆分按照列表进行划分的分区表 people4，将分区 P3 拆分成两个分区 P3_PART1 和 P3_PART2，值'广州'和'桂林'落在第一个分区，剩下的值落在第二个分区。

```
ALTER TABLE people4  SPLIT PARTITION P3  VALUES ('广州','桂林') INTO
(PARTITION P3_PART1,PARTITION P3_PART2);
```



注意：不能拆分按照 Hash 进行划分的两个分区。

10．重命名（RENAME）分区

将分区 P3_PART1 的名字变成 P3_PART11。

```
ALTER TABLE people4 RENAME PARTITION P3_PART1 TO P3_PART11;
```

11．交换分区

Oracle 提供了一种方式，让你在表与表或分区与分区之间进行数据迁移，需要注意的

是，这仅仅是完成表中数据的迁移。

```
ALTER TABLE people4 EXCHANGE PARTITION P3_PART11 WITH TABLE t1
```

12．删除（DROP）分区

删除分区表中指定的分区。

```
ALTER TABLE people4 DROP PARTITION P3_PART11;
```



本章总结

本章主要介绍了表（分区表）、SQL 语句、LOB 类型、用户自定义数据类型。对于本章的每个例子，希望大家能够进行试验，只有试验才能够让大家加深对概念的理解。对于本章涉及的代码，实用性比较强，只需模仿一下，再稍加修改，就可以应用到实际工作中，笔者认为这也是学习软件开发的一种方法。学习开发的其中一种方法就是，模拟别人的示例代码，理解代码的大致原理（不要拘泥于小节），等在实际工作中需要的时候，把代码复制过去，边查参考资料边修改，最终就变成了自己的代码。

第 17 章　构筑高速运行的 SQL 语句

本章重点介绍 SQL 语句的性能调整，在此之前，我们将先对数据库性能调整进行概述。

17.1　数据库性能调整简介

数据库性能调整是 DBA 必备的技能之一，也是开发人员最关注的话题。Oracle 从诞生的那天起，以其高性能赢得大家的青睐。但是，早期的 Oracle 性能调整比较复杂，许多工作需要 DBA 手工完成。伴随着 Oracle 新版本的推出，数据库的性能调整越来越智能化。

数据库性能调整包括实例调整（Instance Tuning）和 SQL 语句调整（SQL Tuning）。两者相互联系和相互影响。如果实例出现性能问题，无论你如何优化 SQL 语句，也不能使 SQL 语句高速运行；反之，即使实例的调整已经到位，一条效率低下的 SQL 语句也足以使整个数据库性能大大降低。

17.2　实例调整

实例调整的重点应该放在数据库的设计上。一旦数据库设计付诸实施，我们将很难在生产环境中改变数据库的设计。数据库的设计能够避免日后的性能瓶颈。实例调整涉及内存调整、I/O 分布和操作系统调整等。实例调整针对的是整个实例的表现，而不是针对单条 SQL 语句。

实例调整的过程如图 17-1 所示。



图 17-1　实例调整的过程



17.2.1 自动工作负载库（Automatic Workload Repository，AWR）

自动工作负载库是 Oracle 公司提供的一个工具，其自动收集、处理、维护性能相关的统计信息。这些统计信息可以协助我们找出 Oracle 的性能瓶颈。

1．AWR 的启用

在默认情况下，Oracle 启用数据库统计收集这项功能（即启用 AWR）。是否启用 AWR 由初始化参数 STATISTICS_LEVEL 控制。通过 SHOW PARAMETER 命令可以显示 STATISTICS_LEVEL 的当前值：

```
SQL> SHOW PARAMETER STATISTICS_LEVEL
```

SQL 语句的执行结果是：

```
NAME                                 TYPE        VALUE
------------------------------------   ----------- ---------------
statistics_level                      string      TYPICAL
```

如果 STATISTICS_LEVEL 的值为 TYPICAL 或者 ALL，表示启用 AWR；如果 STATISTICS_LEVEL 的值为 BASIC，表示禁用 AWR。

2．快照（SNAPSHOT）

每隔一小时，内存监控进程（MMON）自动地采集一次统计信息，并把这些信息存放到负载库中，一次采样就是一个快照。为了节省空间，采集的数据在 7 天后自动清除。快照的频率和保留时间可以由用户修改。

用户也可以使用下面的命令手工采样（手工生成快照）：

```
BEGIN
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
END;
```

通过查询视图 DBA_HIST_SNAPSHOT，可以知道系统中产生了哪些快照。

3．产生 AWR 报告

Oracle 可以产生两种类型的 AWR 报告：文本格式和 HTML 格式。HTML 格式的报告界面更加友好。AWR 报告用于显示两个快照或者两个时间点之间捕捉到的数据。AWR 报告其实就是一张数据库健康体检表，它显示了数据库健康的各项指标。

有针对整个数据库的 AWR 报告，有针对某个实例的 AWR 报告（在集群环境），有针对单条 SQL 语句的 AWR 报告。通过运行 Oracle 自带的 SQL 脚本产生 AWR 报告，只是产生不同的 AWR 报告，需要运行不同的脚本。

■ 产生整个数据库的 AWR 报告，运行脚本 awrrpt.sql。

```
@$ORACLE_HOME/rdbms/admin/awrrpt.sql
```

■ 产生某个实例的 AWR 报告，运行脚本 awrrpti.sql。

```
@$ORACLE_HOME/rdbms/admin/awrrpti.sql
```

■ 产生某条 SQL 语句的 AWR 报告，运行脚本 awrsqrpt.sql。

```
@$ORACLE_HOME/rdbms/admin/awrsqrpt.sql
```



注： $ORACLE_HOME 代表 Oracle 的主目录。

示例：产生整个数据库的 AWR 报告。本例中，Oracle 的主目录是 C:\app\Administrator\product\11.1.0\db_3。

运行脚本 awrrpt.sql。

```
SQL> @C:\app\Administrator\product\11.1.0\db_3\rdbms\admin\awrrpt.sql
```


选择报告的类型。

```
输入 report_type 的值:  html
```

本例中选择 HTML。

列出最近两天产生的快照（主要是获得快照 ID）

```
输入 num_days 的值:  2
Instance     DB Name       Snap Id    Snap Started    Level
------------ ------------ --------- ------------------ -----
zdb          ZDB             386 23 10 月 2009 09:08     1
                             400 24 10 月 2009 09:58     1
                             401 24 10 月 2009 11:00     1
                             402 24 10 月 2009 11:03     1
```


指定起始快照的 ID 和结束快照的 ID。

```
Specify the Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
输入 begin_snap 的值:  400
Begin Snapshot Id specified: 400

输入 end_snap 的值:  402
End   Snapshot Id specified: 402
```

本例中起始快照的 ID 是 400，结束快照的 ID 是 401。

指定报告的名字。

```
输入 report_name 的值:  myreport.html
```



17.2.2 性能监测器（Automatic Database Diagnostic Monitor，ADDM）

性能监控器（ADDM）是 Oracle 提供的强大工具，用于对 Oracle 的性能进行诊断。自动工作负载（AWR）是性能检测器的基础，在使用时，ADDM 对 AWR 收集的数据进行分析，找出性能问题的根本原因，并自动提出问题的解决之道。与此同时，ADDM 也会标识出没有问题的地方，以此消除我们的猜疑。ADDM 可以诊断的问题主要有：

- CPU 的瓶颈
- 内存问题
- I/O 问题
- 高耗能 SQL
- PL/SQL 或 Java 引起的性能问题
- 与 RAC 相关的性能问题
- 数据库配置问题
- 并发性问题
- 热点对象的问题

1．ADDM 的启用

ADDM 是否启用由两个初始化参数 CONTROL_MANAGEMENT_PACK_ACCESS 和 STATISTICS_LEVEL 共同控制。在默认情况下，ADDM 被 Oracle 自动启用。

- CONTROL_MANAGEMENT_PACK_ACCESS 的值为 DIAGNOSTIC 或者 DIAGNOSTIC+TUNING 都将启用 ADDM；如果值为 NONE，将禁用 ADDM。
- 如果 STATISTICS_LEVEL 的值为 TYPICAL 或者 ALL，将启用 ADDM；如果值为 BASIC，则会禁用 ADDM。

可以使用 SHOW PARAMETER 命令显示 CONTROL_MANAGEMENT_PACK_ACCESS 的当前值。

```
SQL> SHOW PARAMETER CONTROL_MANAGEMENT_PACK_ACCESS;
```

SQL 语句的执行结果是：

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------------
control_management_pack_access       string      DIAGNOSTIC+TUNING
```

此外，启用 ADDM 时，还有另外一个参数 DBIO_EXPECTED，I/O 性能的分析在部分程度上依赖于该参数。这个参数的含义是读取一个数据块所花费的平均时间（以微秒为单位）。

2．查看 ADDM 报告

查看 ADDM 报告需要经历两个过程：

- 创建并执行“分析任务”
- 查看报告

查看 ADDM 报告的详细过程说明如下。

STEP 01 查看快照信息。

```
SQL> SELECT * FROM dba_hist_snapshot ORDER BY snap_id;
```

SQL 语句的执行结果是：

```
SNAP_ID DBID       INSTANCE_NUMBER STARTUP_TIME
-------     ----------  ---------------            ----------------
400     2235959226   1              24-10月-09 09.47.06.000 上午
401     2235959226   1              24-10月-09 09.47.06.000 上午
402     2235959226   1              24-10月-09 09.47.06.000 上午
403     2235959226   1              24-10月-09 09.47.06.000 上午
404     2235959226   1              24-10月-09 09.47.06.000 上午
405     2235959226   1              24-10月-09 09.47.06.000 上午
406     2235959226   1              24-10月-09 09.47.06.000 上午
```

这里主要想得到快照的 ID。

STEP 02 创建并执行“分析任务”。

```
VAR taskname VARCHAR2(100);--定义绑定变量
BEGIN
:taskname := 'the whole database analyze';--指定任务的名字
DBMS_ADDM.ANALYZE_DB(:taskname, 404, 406);
END;
```

其中，taskname 是任务的名字。404 及 406 是快照的 ID，404 表示分析的起点，406 表示分析的终点。DBMS_ADDM.ANALYZE_DB 用于分析整个数据库；DBMS_ADDM.ANALYZE_INST 用于分析指定的实例；DBMS_ADDM.ANALYZE_PARTIAL 用于分析数据库的一部分。

STEP 03 查看报告。

```
SET LONG 100000--
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:taskname) FROM DUAL;
```

其中，taskname 是前面步骤中定义的绑定变量。DBMS_ADDM.GET_REPORT 的返回值是 CLOB 类型，因此，需要对输出进行格式化。

Oracle 推荐使用企业管理（OEM）运行 ADDM，企业管理器是图形界面工具，界面友好，方便查看 ADDM 报告。



17.3 SQL 语句调整

SQL 语句调整是本章的重点，也是数据库性能调整中技巧性最强的部分。SQL 语句调整针对的是特定的 SQL 语句，而不是整个数据库。在介绍 SQL 语句调整之前，我们先介绍索引。

17.3.1 数据查询的加速器——索引

本节将要介绍各种类型的索引。需要记住的是，索引是一把双刃剑！

1．索引简介

索引是一种与表相关的、可选的结构，用于加速对表的访问。当用户访问表中的数据时，Oracle 从索引中找到数据的指针，然后根据指针指向的位置从表中直接取得数据。索引可以减少磁盘 I/O，加快数据访问速度。

索引由 Oracle 自动维护，无须用户的干预。索引也是模式对象。

唯一性索引

索引有唯一性索引（Unique Indexes）和非唯一性索引（Nonunique Indexes），唯一性索引能够确保索引列不会出现重复值。

复合索引

通常索引都是由一列组成，而复合索引由多个索引列组成。

索引的分类

- 二叉树索引或者叫 B 树索引（B-tree Indexes），B 树索引是使用得最多的一种索引。在默认情况下，我们创建的索引都是 B 树索引。B 树索引基于二叉树原理，有关二叉树的原理，请大家参考与数据结构相关的书籍。
- 二叉树聚簇索引（B-tree Cluster Indexes），主要用于聚簇。
- 哈希聚簇索引（Hash Cluster Indexes），主要用于哈希（Hash）聚簇。
- 反向索引（Reverse Key Indexes），反向索引也属于 B 树索引，它把索引值按字节反转过来。
- 位图索引（Bitmap Indexes），指通过位图对索引进行管理，位图索引适合唯一值很少的列，也就是重复值很多的列。
- 位图连接索引（Bitmap Join Indexes），用于两个表的连接。
- 基于函数的索引（Function-Based Indexes），如果在 SQL 语句的 WHERE 子句中经常用到函数或者表达式，则可以创建基于函数的索引。

2．创建索引

■ 创建二叉树索引（普通索引）

```
CREATE INDEX ind_name4  ON ct(name);
```

在表 ct 的列 name 上创建索引 ind_name4。

■ 创建唯一性索引

```
CREATE UNIQUE INDEX ind_name
ON ct(name)
```

```
    TABLESPACE BIGTBS_01;
```

■ 创建位图索引

```
    CREATE BITMAP INDEX ind_name5  ON ct2(name);
```

■ 创建反向索引

```
    CREATE INDEX ind_name3 ON ct(name) REVERSE;
```

■ 创建函数索引

```
    CREATE INDEX ind_name2 ON ct(UPPER(name));
```

3．把索引移动到另外一个表空间

由于空间的调整，需要把索引从一个表空间移动到另外一个表空间，Oracle 的 ALTER INDEX ... REBUILD TABLESPACE…命令提供了索引移动的功能。

示例：移动索引 INDEX_PS。

移动索引以前，索引 INDEX_PS 在表空间 SYSTEM 中。

```
    SQL> SELECT index_name,tablespace_name FROM dba_indexes WHERE index_name=
'INDEX_PS';
    INDEX_NAME                     TABLESPACE_NAME
    ------------------------------ ------------------------------
    INDEX_PS                       SYSTEM
```

索引 INDEX_PS 存在于表空间 SYSTEM 中。

执行下面的命令，把索引从表空间 SYSTEM 移动到另外一个表空间 product。

```
    ALTER INDEX INDEX_PS REBUILD TABLESPACE product;
```

把索引 INDEX_PS 移动到 product 表空间。

确认索引是否已经成功移动到另外一个表空间。

```
    SQL> SELECT index_name,tablespace_name FROM dba_indexes WHERE index_name=
'INDEX_PS';
```

SQL 语句的执行结果是：

```
    INDEX_NAME                     TABLESPACE_NAME
    ------------------------------ ------------------------------
    INDEX_PS                       PRODUCT
```

由此可以看出，索引 INDEX_PS 已经成功地从表空间 SYSTEM 移动到表空间 PRODUCT。

4．为何要重建索引

如果索引因为某些原因无效或者因为很长时间没有维护而产生过多的索引碎片（Index Fragment），需要通过重建索引来消除索引碎片。何时需要重建索引，可以利用下面的过程

进行判断。

STEP 01 查询数据库中有哪些索引。

```
SQL>SELECT OWNER,INDEX_NAME FROM DBA_INDEXES;
```

SQL 语句的执行结果是：

```
OWNER                          INDEX_NAME
------------------------------ ------------------------------
ITME                           PK1
ITME                           INDF5
ITME                           INDF4
ITME                           KK
ITME                           KT2
ITME                           PK11
…
```

本文将以索引 INDF5 为例。

STEP 02 对索引 INDF5 进行分析。

```
SQL> ANALYZE INDEX INDF5 VALIDATE STRUCTURE;
Index analyzed
```

STEP 03 从视图 INDEX_STATS 中获得索引 INDF5 的统计信息。

```
SQL>SELECT HEIGHT,(DEL_LF_ROWS_LEN/LF_ROWS_LEN)*100 FROM INDEX_STATS WHERE
NAME='INDF5';
```

SQL 语句的执行结果是：

```
 HEIGHT (DEL_LF_ROWS_LEN/LF_ROWS_LEN)*
---------- ------------------------------
         1   23.1
```

其中，字典 INDEX_STATS 表示存放索引的统计信息；列 DEL_LF_ROWS 表示索引删除行数；列 LF_ROWS 表示索引总行数；列 HEIGHT 表示二叉树中从根块到叶块的层次（深度）。

如果满足下面其中一个条件，则要考虑重建索引：

- (DEL_LF_ROWS_LEN/LF_ROWS_LEN)*100 的值大于 20
- HEIGHT 的值大于 4（说明二叉树的层次太多）

STEP 04 使用 ALTER INDEX ... REBUILD 命令重建索引 INDF5。

```
SQL> ALTER INDEX INDF5 REBUILD;
```

5．消除索引的副作用——删除多余的索引

如果表上存在过多的索引，可能会降低 DML 语句的执行速度，需要使用 DROP INDEX 命令删除索引。用户可以删除自己创建的索引。如果要删除其他用户的索引，必须具有 DROP ANY INDEX 系统权限。

删除索引 ind_name3。

```
SQL> DROP INDEX ind_name3;
```

6．查看索引信息

数据字典 DBA_INDEXES 用于列出数据库中索引的信息，从 DBA_INDEXES 中，可以知道数据库中有哪些索引、每个索引的名字、索引的类型、索引对应的基表等信息。

示例：查询数据库中有哪些索引。

```
SQL> SELECT index_name,index_type,table_name
FROM dba_indexes;
```

17.3.2 优化器（Optimizer）

优化器用于产生执行计划（Execution Plan）。产生执行计划时，优化器会对表达式和条件进行评估，它综合各方面的因素（如统计信息、表的连接顺序、优化目标、访问路径等），然后生成执行效率最高的执行计划。

17.3.3 SQL 语句执行的大致过程

SQL 语句的执行过程如图 17-2 所示。当用户向数据库发送一条 SQL 语句后，Oracle 创建一个游标，并查询共享池，如果 SQL 语句存在共享池中，将跳过解析，直接选择共享池中的 SQL 语句；如果 SQL 语句不存在共享池中，Oracle 将对 SQL 语句进行解析，并把解析后的 SQL 语句放入共享池中，然后 Oracle 从多个执行计划中选择一个执行计划（解析时会产生多个执行计划），并进行变量绑定，变量绑定完成后，Oracle 将执行该 SQL 语句，并把执行的结果返回给用户（如果是查询），最后，Oracle 关闭游标。

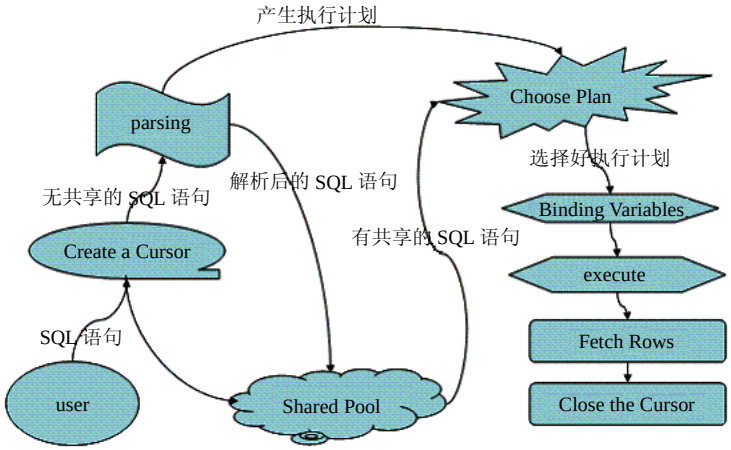


图 17-2　SQL 语句的执行过程

SQL 语句解析将做下面的工作：

- 对 SQL 语句进行语法检查。
- 查询数据字典，校验表和列的定义。
- 权限检查。
- 产生执行计划（会产生多个执行计划）。
- 如果是分布式相关的语句，Oracle 会把 SQL 的一部分发送到远程的数据库进行验证。



17.3.4 SQL 语句的优化过程

本节介绍 SQL 语句的性能优化过程，并引入和 SQL 语句优化相关的一些知识。SQL 语句的优化过程如图 17-3 所示。

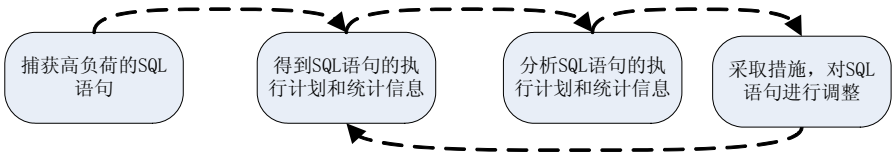


图 17-3　SQL 语句的优化过程

1．找出高负荷 SQL

如果数据库运行缓慢，或者应用程序的某些功能出现停滞现象，可能是高负荷 SQL 语句（High-Load SQL）阻塞了应用程序的执行，这时候，我们要抓出高负荷的 SQL 语句。所谓“高负荷 SQL 语句”是指那些性能低下，且消耗过多系统资源的 SQL 语句。下面的方法（工具）可以用于找出高负荷 SQL 语句：

- 性能检测器 ADDM
- 自动化 SQL 调整（Automatic SQL Tuning）
- 自动工作负载库 AWR
- 查看视图 V$SQL
- 来自客户的报告
- SQL 跟踪工具（SQL Trace）

2．查看执行计划

一旦获得高负荷的 SQL 语句，就需要对该 SQL 语句进行调整，SQL 语句调整的工具比较多，这些工具使 SQL 语句的调整越来越智能化，简单化！这些工具是：

- SQL Trace（SQL 跟踪）提供单条 SQL 语句的性能信息（包括统计信息）。
- AUTOTRACE（自动跟踪）是 SQL*Plus 的一个功能，也是用于产生某条 SQL 语句的性能信息。
- SQL Access Advisor（SQL 访问指导）通过建议要创建、删除或保留的索引、物化视图、物化视图日志或分区来确定并帮助解决与 SQL 语句执行相关的性能问题。
- SQL Tuning Advisor（SQL 调整指导）可以为 SQL 的优化提供建议，使 SQL 的调

整简单化。

- SQL Performance Analyzer（SQL 性能分析器）可用于预测数据库环境改变对 SQL 执行造成的负面影响。

有 SQL 调整经验的朋友都知道，SQL 调整是一个重复的过程。从使用方便的角度考虑，笔者个人喜欢使用 AUTOTRACE（因为 SQL Trace 使用比较麻烦，SQL Access Advisor 和 SQL Tuning Advisor 通常又依赖于运行在服务器端的数据库控件）。从智能化的角度考虑，可以使用 SQL Access Advisor 和 SQL Tuning Advisor。本书重点介绍 AUTOTRACE。

📖 **配置** AUTOTRACE

在**第一次**使用 AUTOTRACE 时，需要配置 AUTOTRACE。AUTOTRACE 要求创建执行计划表（PLAN_TABLE）和角色 PLUSTRACE，并把角色 PLUSTRACE 授予要使用 AUTOTRACE 的用户（本例中是 ITME）。详细过程说明如下。



注：本例中的 ORACLE_HOME 是 C:\app\Administrator\product\11.1.0\db_3。

STEP 01 创建执行计划表 PLAN_TABLE。

```
SQL> CONNECT itme
SQL> @$ORACLE_HOME\rdbms\admin\utlxplan.sql
```

STEP 02 创建角色 PLUSTRACE。

```
SQL> CONNECT / AS SYSDBA
SQL> @$ORACLE_HOME\sqlplus\admin\plustrce.sql
```

STEP 03 把角色 PLUSTRACE 授予用户 itme。

```
SQL> CONNECT / AS SYSDBA
SQL> GRANT PLUSTRACE TO itme;
```

📖 **运行** AUTOTRACE（**获得** SQL **语句的执行计划**）

每次用 SQL*Plus 登录数据库以后，需要打开 AUTOTRACE，接下来执行的每条 SQL 都会**自动显示**执行计划，这是笔者最喜欢 AUTOTRACE 的原因。打开的 AUTOTRACE 只对本次会话生效，下次启动 SQL*Plus，还需要重新打开 AUTOTRACE。AUTOTRACE 的选项开关如下所示：

- SET AUTOT[RACE] OFF 关闭 AUTOTRACE。
- SET AUTOT[RACE] ON 打开 AUTOTRACE，显示 SQL 语句的执行计划和统计信息，还显示 SQL 语句的执行结果。
- SET AUTOT[RACE] TRACEONLY 打开 AUTOTRACE，仅显示 AUTOTRACE 信息，不显示 SQL 语句的执行结果。
- SET AUTOT[RACE] ON EXPLAIN 打开 AUTOTRACE，仅显示 SQL 语句的执行计划。
- SET AUTOT[RACE] ON STATISTICS 打开 AUTOTRACE，仅显示 SQL 语句的统计

信息。

AUTOTRACE 的使用过程说明如下。

STEP 01 打开 AUTOTRACE（可使用 AUTOTRACE 的各种选项开关）。

```
SET AUTOTRACE TRACEONLY EXPLAIN STATISTICS;
```

这是笔者进行 SQL 调整时用得最多的 AUTOTRACE 选项组合。它只显示执行计划和统计信息，不显示 SQL 的返回结果。



注：执行计划涉及的表和数据，请参考光盘\script\执行计划案例脚本\。

STEP 02 执行有问题的 SQL 语句，我们将自动获得有问题的 SQL 语句的执行计划。

```
SQL> SELECT a.op_no, a.fnm, b.menu_id, b.gwa_code, b.tr_code, b.button_flg
  2    FROM tb_user a, tb_user_grant b
  3   WHERE a.op_no = b.op_no;
已选择 256 行。

执行计划
----------------------------------------------------------
Plan hash value: 3384233913

------------------------------------------------------------------------------

| Id  | Operation          | Name         | Rows  | Bytes | Cost (%CPU)| Time     |

------------------------------------------------------------------------------

|   0 | SELECT STATEMENT   |              |   256 |  9984 |     7  (15)| 00:001 |

|*  1 |  HASH JOIN         |              |   256 |  9984 |     7  (15)| 00:001 |

|   2 |   TABLE ACCESS FULL| TB_USER      |   113 |  1695 |    3   (0)| 00:001 |

|   3 |   TABLE ACCESS FULL| TB_USER_GRANT |   257 |  6168 |     3   (0)| 00:001 |

------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("A"."OP_NO"="B"."OP_NO")

统计信息
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         32  consistent gets
```

```
          0  physical reads
          0  redo size
       7748  bytes sent via SQL*Net to client
        603  bytes received via SQL*Net FROM client
         19  SQL*Net roundtrips to/FROM client
          0  sorts (memory)
          0  sorts (disk)
        256  rows processed
SQL>
```

上面的 SQL 报告分成两部分：第一部分显示 SQL 语句的**执行计划**；第二部分显示 SQL 语句的**统计信息**。

AUTOTRACE 打开之后（只需打开一次），我们可以执行其他 SQL 语句，获得其他 SQL 语句的执行计划。

3．执行计划的阅读

本节需要了解执行计划的阅读方法和执行步骤（访问路径）的含义。

📖 阅读方法

执行计划是一个树状结构。每一行就是一个执行步骤（操作），每个步骤有编号（ID），每个步骤会返回数据行（Rows），这些数据行会被下一步进行处理，每个步骤还会显示处理了多少字节（Bytes）的数据及该步骤的执行代价（Cost，这个值是相对值）和消耗的时间（以秒为单位）。Oracle 最先从叶子节点（FIRST　GRANDCHILD）开始执行，然后执行到上一层（FIRST　CHILD、SECOND CHILD），层层递进，最后执行到顶层（PARENT）。如果同一层中有并列的多个步骤（如 FIRST　CHILD、SECOND CHILD），最上面的步骤（FIRST　CHILD）首先被执行。ID 前面有星号（*），表示下面有详细的解释。一个执行计划由多个执行步骤组成。

```
PARENT（父亲）
    FIRST  CHILD （孩子）
        FIRST  GRANDCHILD （孙子）
    SECOND CHILD（孩子）
```

SQL 语句的执行过程说明如下（参考上一节产生的执行计划）：

```
SELECT a.op_no, a.fnm, b.menu_id, b.gwa_code, b.tr_code, b.button_flg
  FROM tb_user a, tb_user_grant b
 WHERE a.op_no = b.op_no;
```

STEP 01 对表 TB_USER 执行全表扫描（TABLE ACCESS FULL），返回 113 条记录，执行代价是 3，执行时间是 00:001 秒。

STEP 02 对表 TB_USER_GRANT 执行全表扫描（TABLE ACCESS FULL），返回 257 条记录。

STEP 03 对两次扫描的数据进行连接（HASH JOIN），共处理 256 条记录。

STEP 04 返回结果集。

访问路径（Access Path）

访问路径是指从数据库中检索数据的方法。访问路径有全表扫描（Full Table Scans）、行 ID 扫描（Rowid Scans）、索引扫描（Index Scans）、聚簇访问（Cluster Access），哈希访问（Hash Access）和采样表扫描（Sample Table Scans）。

全表扫描：Oracle 顺序地读取分配给表的每个数据块，直到读到表的高水线位线（High Water Mark，HWM，用于标识表的最后一个数据块）。说白了，全表扫描会读取表中所有的行。

索引扫描：先通过索引查找到数据对应的 ROWID 值，然后根据 ROWID 直接从表中得到具体的数据。一个 ROWID 唯一地标识一行数据。

行 ID 扫描：通过行 ID 直接获得需要的数据，这是访问数据最快的一种方式。行 ROWID 定义了一行数据所在的数据文件、数据块以及行在该块中的位置。通过 ROWID 访问数据，需要首先获取被访问行的 ROWID。

聚簇访问：访问存储在聚簇中的表。在聚簇中，具有相同聚簇值（Cluster Key Value）的行被物理地存储在一起。聚簇访问首先扫描聚簇索引，从聚簇索引中获得 ROWID，然后根据 ROWID 定位需要访问的数据。

哈希访问：哈希扫描用于在哈希聚簇中定位数据行，在哈希聚簇中，具有相同哈希值的行被存放在相同的数据块中。进行哈希扫描时，Oracle 应用哈希函数得到哈希值，然后根据哈希值扫描数据块。

采样表扫描：采样表扫描指按照一定的百分比对表的数据进行采样。如：

```
SQL> SELECT *
  2  FROM tb_user_grant SAMPLE BLOCK (10);
已选择 257 行。
执行计划
----------------------------------------------------------
Plan hash value: 3704821630
------------------------------------------------------------------------------
| Id  | Operation         | Name         | Rows  | Bytes | Cost (%CPU)| Time
|
|   0 | SELECT STATEMENT    |              |    26 |   832 |     2   (0)|
00:00:01 |
|   1 |  TABLE ACCESS SAMPLE| TB_USER_GRANT |    26 |   832 |  2   (0)|
00:00:01 |
```

上述语句访问表 tb_user_grant 中 10%的数据。

连接（Join）方法

连接是指从一个或者多个表中检索数据。Oracle 主要有 3 种连接方法：嵌套循环连接（Nested Loop Joins）、哈希连接（Hash Joins）和排序合并连接（Sort Merge Joins）。在 SELECT 语句中，连接方法至关重要，它直接影响查询的性能。

- 嵌套循环连接（Nested Loop Joins）

嵌套循环连接适合处理小量数据。Oracle 首先选择一张表作为连接的驱动表（Driving Table），这张表也被叫做外部表（Outer Table），其他表被当做内部表（Inner Table），内部表也叫被驱动表。对于驱动表中的每行，Oracle 将访问内部表中的所有行。

嵌套循环连接可以快速返回查询结果的前几行，这是因为，它不需要等到全部循环结束再返回结果集，而是不断地返回查询的结果，也就是语句边执行边返回结果。

- 哈希连接（Hash Joins）

哈希连接用于处理大数据量的数据集（特别是一个大表和一个小表的连接）。Oracle 从两个表中选择一个较小的表，按照连接关键字（Join Key），在内存中建立哈希表（因此哈希表中较小的表适合放在内存中），然后，Oracle 扫描另外一个表（大表），从中发现可以被连接的行。

- 排序合并连接（Sort Merge Joins）

排序合并连接的原理是，两个表先按照连接关键字（Join Key）进行排序，再把排好序的两个结果集合并在一起。所以，如果其中一个结果集已经排好序，“排序合并连接”方式的速度将比“哈希连接”方式快。

4．统计信息的解释

通过 AUTOTRACE，我们不仅可以获得 SQL 语句的执行计划，还可以获得 SQL 语句的统计信息。接下来，我们将对 SQL 语句的统计信息进行解释。

0　recursive calls——递归调用的次数。

0　db block gets——当前块（Current Block）被请求的次数，当前块是指未被修改过的块。

32　consistent gets——对块一致性读的请求次数，一致性读涉及读回滚段。

0　physical reads——从磁盘读的块的数量。

0　redo size——产生的回滚数据的大小，单位是字节（Bytes）。

7748　bytes sent via SQL*Net to client——后台进程发送给客户端的信息量大小，单位是字节。

603　bytes received via SQL*Net FROM client——客服端发给数据库服务器的信息量的大小，单位是字节。

19　SQL*Net roundtrips to/FROM client——Oracle Net 发送和接受的信息数量，单位是条。

0　sorts (memory)——内存排序的次数。

0　sorts (disk)——磁盘排序的次数。

256　rows processed——处理数据的行数（通常指返回的记录数）。

5．SQL 优化的目标

有的朋友说，减少一条 SQL 语句的执行时间是我们优化 SQL 语句的目标。对，没错，但是在大部分情况下，我们不能在生产系统中直接执行调试中的 SQL 语句，以获得该语句的执行时间。不过，在不真正执行 SQL 语句的情况下，我们有一些衡量指标，这些指标可以帮助我们衡量优化后的 SQL 语句是否真的比以前的 SQL 语句更加高效。因此，根

据经验，笔者总结 SQL 调整的目标是：

- 减小 consistent gets 的值
- 减小 db block gets 的值
- 减少排序，尤其磁盘排序 sorts（disk）
- 减少递归调用（recursive calls）
- 减小执行代价（cost）

要实现 SQL 调整的目标，我们需要采取诸如加索引、改写 SQL 语句等措施。



知识点索引——执行代价

执行代价与 SQL 语句使用的资源（I/O、CPU、内存等）有关。如果一条 SQL 语句使用的资源较多，我们就说该 SQL 语句的执行代价大。执行代价的大小是衡量 SQL 语句是否高效的标准之一，但是，执行代价是一个相对值。

6．SQL 优化措施（基本准则）

本节将介绍 SQL 语句优化的措施。SQL 语句的优化是一个复杂而循序渐进的过程。在对 SQL 语句实行优化措施之后，我们要查看衡量 SQL 语句的各项指标是否下降。有的优化措施不仅不能加速 SQL 语句的执行，反而使 SQL 语句的执行更加缓慢。进行 SQL 优化时，我们可以采取下面的措施：

- 收集统计信息
- 重构索引
- 重构数据
- 禁用约束和触发器
- 尽量使用相同的 SQL 语句
- 改写 SQL 语句

收集统计信息

统计信息是存放在数据字典中的一系列的数据，它用于精准地描述数据库和数据库中的对象（如表、索引）。统计信息包括：表的统计信息、列的统计信息、索引的统计信息、系统的统计信息（包括 CPU 和 I/O 的）。统计信息能够帮助优化器选择最好的执行计划，所以，统计信息的精确性尤为重要。如果你没有收集统计信息或者很长时间没有重新收集统计信息（旧的统计信息不能准确地反映数据变化），都将影响优化器选择最优的执行计划。

因为数据库中的对象和数据在不断变化，所以我们应该定期收集统计信息。有以下两种方法收集统计信息。

- 自动收集统计信息

在默认情况下，Oracle 启动了统计信息的自动收集功能，如果你禁用了统计信息的自动收集，可以使用下面的命令启用统计信息的自动收集。

```
BEGIN
DBMS_AUTO_TASK_ADMIN.ENABLE(
```

```
client_name => 'auto optimizer stats collection',
operation => NULL,
window_name => NULL);
END;
```

自动收集统计信息会消耗一部分系统资源，如果没有必要可以关闭该功能。关闭自动收集统计信息的命令如下所示：

```
BEGIN
DBMS_AUTO_TASK_ADMIN.DISABLE(
client_name => 'auto optimizer stats collection',
operation => NULL,
window_name => NULL);
END;
```

手动收集统计信息

如果我们不想启用“自动收集统计信息”，可以利用包 DBMS_STATS 来手工收集统计信息。包 DBMS_STATS 包含如下的过程：

■ 收集索引的统计信息

```
EXECUTE DBMS_STATS.GATHER_INDEX_STATS('ITME','KK');
```

上述命令收集用户模式 itme 下的索引 kk。

■ 收集表的统计信息

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS('ITME','STUDENTBOOK');
```

上述命令收集用户 itme 的表 studentbook 的统计信息。

■ 收集用户拥有的所有对象的统计信息

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('ITME');
```

上述命令收集用户 itme 拥有的所有对象的统计信息。

■ 收集数据字典对象的统计信息

```
EXECUTE DBMS_STATS.GATHER_DICTIONARY_STATS;
```

■ 收集数据库中所有对象的统计信息

```
EXECUTE DBMS_STATS.GATHER_DATABASE_STATS;
```

重构索引

重构索引的工作包括以下几方面：

- 删除多余的索引，以加速 DML 语句的执行。如果一个表上的操作是以 DML 为主，考虑删除冗余的 DML。
- 删除组成索引列中无用的列。因为有的列在 WHERE 条件中很少使用或者从未使用，这样的列不仅不利于查询，可能会反而阻碍 DML 语句的执行。
- 改变索引中列的顺序，使索引中列的顺序和 WHERE 条件中列的顺序相同。

- 考虑往索引中添加新的列。
- 重建索引，消除索引中的碎片。



注意：索引不是灵丹妙药。有时候索引却会加重系统的负担。

重构数据

如果数据分布不合理，即使我们再怎么调整 SQL 语句，效果也只是杯水车薪。数据的分布是否合理将直接影响 SQL 的性能。在此，我们主要讨论数据分布的两项技术。

- 数据分布的其中一项技术是分区（Partitioning），理想的分区技术是，将一个大表按照一定的逻辑分成 *n* 块，每块单独存放在一个物理磁盘中，这时候需要 *n* 块磁盘，并在分区表上创建相应的索引。
- 数据分布的另外一项技术是聚簇（Cluster）。如果两个表经常按照某列进行连接（Join），则可以把这两个表放到聚簇中。这项技术使两个表中的数据物理地存放在一起，减少 I/O，加速对两个表的访问。

禁用约束和触发器

使用触发器会消耗系统资源。如果有太多的触发器存在，系统性能势必受到影响，这时候，可以考虑禁用没有必要的触发器。同理，也需要考虑禁用不必要的约束。

尽量使用相同的 SQL 语句

在编码时，应该尽量使用相同的 SQL 语句，使用相同的 SQL 语句可以减少解析时间。我们都知道，解析要做好多工作，这些工作将延长整条 SQL 语句的执行时间。

两条 SQL 语句相同必须满足的条件是：

- 空格、大小写、换行、注释、表名、关键字、值相同。
- 引用的对象必须属于相同的模式。
- 如果使用绑定变量，绑定变量的名字、类型、长度必须相同。

例：两条不同的 SQL 语句（大小写不同）：

```
SELECT * FROM Qmm;
SELECT * FROM qmm;
```

改写 SQL 语句

在对 SQL 语句的外围环境（如索引、统计信息、数据分布等）进行调整后，如果还不能获得预期的执行速度，就需要对原来的 SQL 语句进行改写。改写 SQL 语句是一项很有挑战性的工作，它不仅需要实现与原来的 SQL 相同的功能，新的 SQL 语句还必须能够高速运行（相对原来的 SQL 语句）。改写 SQL 语句有时候需要使 SQL 语句的结构发生彻底的改变，而不仅仅局限于 SQL 语句的小修小补。改写 SQL 语句时，可以参考下面的原则：

- 要从执行计划中仔细查看、分析 SQL 语句中的每个表是否被高效地访问。
- 避免在 WHERE 子句中的列上使用函数，这样会使列上的索引失效。如果真的想在列上使用函数，则需要创建函数索引。

例：

在列上使用函数，该列 order_name 上的索引失效。

```
SELECT * FROM T_ORDER
WHERE UPPER(order_name)='PC';
```

如果想在列 order_name 上利用函数，必须创建函数索引 indf3。

```
CREATE INDEX indf3 on T_ORDER(UPPER(order_name));
```

- 尽量使用等价连接。
- 排序列上应该建立索引。
- 保持排序列和索引列次序一致。
- 尽量保持 WHERE 子句中列的顺序和索引列的顺序一致。
- 尽量少用嵌套查询。如果非要使用，请用 NOT EXIST 代替 NOT IN 子句。
- 用多表连接代替 EXISTS 子句。
- 选择适合的连接（join）。如果关注响应时间，可以使用嵌套循环连接（Nested Loop Joins）；如果关注吞吐量，且数据源已经排序，可以使用排序合并连接（Sort Merge Joins）；如果是一个小表和大表的连接，考虑使用哈希连接（Hash Joins）。
- 在连接（Joins）中，注意表的连接顺序。要选择好驱动表（Outer Table）和被驱动表（Inner Table）。通常，驱动表是由过滤条件限制返回记录最少的那张表，而不是数据量最少的那张表。
- 大量的排序操作影响系统性能，所以尽量减少排序操作。GROUP BY、ORDER BY、ROLLUP、DISTINCT 等都会产生排序。少用 DISTINCT，用 EXISTS 代替 DISTINCT。
- 避免使用数据类型自动转换功能，自动数据类型转换会增加额外的系统开销，还会使索引无效（也就是不能利用存在的索引）。
- 启用并行执行。并行执行机制使多个服务器进程并行地执行一条 SQL 语句，这是 Oracle 的强大之处。
- 应尽量使用 VARCHAR2 类型代替 CHAR 类型，这样可以节省磁盘空间，减少兼容带来的问题。
- 连接符 OR、IN、AND 以及＝、<=、>=等前后加上一个空格。这是一种良好的书写习惯，防止产生两条不同的 SQL 语句（而实际上它们是同一条 SQL 语句，只是不好的书写习惯让 Oracle 把它们当成两条 SQL 语句）。
- WHERE 子句中尽量使用变量，这样可以减少语句的解析时间。

例：

查询语句 1：

```
SELECT COUNT(*) FROM T_ORDER
WHERE ORDER_NO>10;
```

查询语句 2：

```
SELECT COUNT(*) FROM T_ORDER
WHERE ORDER_NO>5;
```

以上两条 SQL 语句被 Oracle 当成两条独立的语句，会耗费额外的解析时间。解决方

法是使用变量代替常量。

```
SELECT COUNT(*) FROM T_ORDER
WHERE ORDER_NO>:a;
```

- Oracle 的关键字（如内置函数名、SQL 保留字大写等）要大写，其他的字符都用小写。这样做的目的也是为了防止产生两条不同的 SQL 语句。

LIKE 用于模糊查询，如果百分号前面有字符，将会用到索引。

```
SELECT * FROM tb_user WHERE fnm like 'Zhang%';
```

如果百分号在字符串的中间，将会利用索引。

```
SELECT * FROM tb_user WHERE fnm like 'Zh%g';
```

如果百分号前面和后面都有字符，不会用到索引。

```
SELECT * FROM tb_user WHERE fnm like '%Zhang%';
```

如果只有百分号后面有字符，将不会用到索引。

```
SELECT * FROM tb_user WHERE fnm like '%Zhang';
```

- 要避免使用结构复杂的视图，结构复杂的视图隐含查询性能问题。
- 如果可以，把查询结果放在一个中间表中，程序需要时，直接查询中间表，这是改善查询最有效的手段之一。
- 考虑使用物化视图（Materialized Views）代替结构复杂的查询。
- 减少对表的扫描，可以把多个扫描合并成一个扫描。

例：

查询语句 1 对表 t_order 进行了 1 次扫描。

```
SELECT COUNT (*)
FROM t_order
WHERE ORDER_NO < 1000;
```

查询语句 2 对表 t_order 进行了 1 次扫描。

```
SELECT COUNT (*)
FROM t_order
WHERE ORDER_NO BETWEEN 1000 AND 3000;
```

查询语句 3 对表 t_order 进行了 1 次扫描。

```
SELECT COUNT (*)
FROM t_order
WHERE ORDER_NO>3000;
```

以上 3 条 SQL 语句共对表 t_order 扫描了 3 次。

解决方法是合并 SQL，把 3 次扫描变成 1 次扫描。

```
SELECT COUNT (CASE WHEN ORDER_NO < 1000
THEN 1 ELSE null END) cnt1,
```

```
    COUNT (CASE WHEN ORDER_NO BETWEEN 1001 AND 3000
    THEN 1 ELSE null END) cnt2,
    COUNT (CASE WHEN ORDER_NO > 3000
    THEN 1 ELSE null END) cnt3
    FROM t_order;
```

- 使用带 RETURNING 子句的 INSERT、UPDATE、DELETE 语句完成选择数据和修改数据两个功能，减少对数据库的调用。
- 尽量让一条 SQL 语句完成更多的功能。
- 在 WHERE 子句中使用 IS NULL 或者 NOT NULL 将使索引失效。

例：在表 tb_user_grant 的列 menu_id 上创建索引。

```
    CREATE INDEX kt2 ON tb_user_grant(menu_id);
```

虽然列上存在索引，但 IS NULL 使查询 SQL 语句不能够利用索引，在表 tb_user_grant 上执行的仍然是全表扫描。

```
    SQL> SELECT * FROM tb_user_grant WHERE  menu_id IS NULL and op_no='5454';
    执行计划
    ----------------------------------------------------------
    Plan hash value: 3624973521
    ------------------------------------------------------------------------------
    | Id  | Operation       | Name        | Rows  | Bytes | Cost (%CPU)| Time  |
    ------------------------------------------------------------------------------
    |   0 | SELECT STATEMENT  |               |    56 |  1344 |   171   (1)|
00:00:03 |
    |*  1 |  TABLE ACCESS FULL| TB_USER_GRANT |    56 |  1344 |   171   (1)|
00:00:03
```

- 不要在 WHERE 子句中使用连接符（||），连接符将忽略索引。

17.3.5 提示（Hint）

提示（Hint）是嵌入到 SQL 语句中的一些指令，用于改变 SQL 语句的执行计划。如有一条 SQL 语句执行的是索引扫描，我们可以使用“提示”强迫它执行全表扫描（尽管没有必要）。

1. 提示的分类

有几大类的提示：

- 有关于优化目标的提示
- 有关于访问路径的提示
- 有关于查询转换的提示
- 有关于连接操作的提示
- 有关于连接顺序的提示

➢ 有关于并行执行的提示

2．提示的语法

提示的语法是/*+ hint(argument1 argument2) */，**提示**以/*+开始，以*/结束，中间是**提示**指令。argument1 表示提示的第一个参数，argument2 表示提示的第二个参数。当优化器遇到**提示**时，将按照“**提示**”的意图执行 SQL 语句。在指定**提示**时，如果出现错误，优化器将忽略该**提示**。提示中不能出现模式名。

下面的查询语句会利用到索引 INDF5。

```
SQL> SELECT * FROM T_ORDER
  2  WHERE order_name='PC';
执行计划
----------------------------------------------------------
Plan hash value: 748836800
-------------------------------------------------------------------------------
| Id  | Operation             | Name    | Rows  | Bytes | Cost (%CPU)| Time|

-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |         |     1 |    37 |     1   (0)| 00:00:01 |

|   1 |  TABLE ACCESS BY INDEX ROWID| T_ORDER |  1 |    37 |     1   (0)|
00:00:01

|*  2 |   INDEX RANGE SCAN          | INDF5   |     1 |       |     1   (0)|
00:00:01|
```

在 SQL 语句中加入提示 FULL，表示要对表 T_ORDER 执行全表扫描。

```
SQL> SELECT /*+FULL(t)*/ * FROM T_ORDER t
  2  WHERE order_name='PC';
执行计划
----------------------------------------------------------
Plan hash value: 2317089884
-------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     1 |    37 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T_ORDER |     1 |    37 |     2   (0)| 00:00:01 |
```

提示 FULL 改变了 SQL 语句的执行计划，使 Oracle 对表 T_ORDER 执行全表扫描。

3．常用提示

在 Oracle 中，有许多提示，我们将节选部分常用的提示进行介绍。

🕮 ALL_ROWS

ALL_ROWS 表示以吞吐量为优化目标。

```
SELECT /*+ALL_ROWS*/  *  FROM T_ORDER;
```

📖 FIRST_ROWS(n)

FIRST_ROWS(*n*)表示选择基于开销的优化方法，以响应时间为优化目标，*n* 表示数据行数。

```
SELECT /*+FIRST_ROWS*/  *  FROM T_ORDER;
```

📖 INDEX

该提示使 Oracle 执行索引扫描。

```
SELECT /*+ INDEX (t kt2)*/ op_no
FROM tb_user_grant t
WHERE t.menu_id like '027%';
```

本例使 Oracle 执行表 tb_user_grant（别名是 t）上的索引扫描，索引的名字是 kt2。

📖 USE_MERGE

提示 USE_MERGE 用于指示两个表使用“排序合并连接（Sort Merge Joins）”方式进行连接（JOIN）。

```
SELECT  /*+ USE_MERGE(a b) */ a.op_no, a.fnm, b.menu_id, b.gwa_code,
b.tr_code
FROM tb_user a, tb_user_grant b
WHERE a.op_no = b.op_no;
```

本例中，表 tb_user（别名 a）和表 tb_user_grant（别名 b）使用 Sort Merge Joins 方式进行连接。

📖 USE_HASH

提示 USE_HASH 用于指示两个表使用“哈希连接（Hash Joins）”方式进行连接（JOIN）。

```
SELECT  /*+ USE_HASH(a b) */ a.op_no, a.fnm, b.menu_id, b.gwa_code,
b.tr_code
FROM tb_user a, tb_user_grant b
WHERE a.op_no = b.op_no;
```

本例中，表 tb_user（别名 a）和表 tb_user_grant（别名 b）使用 Hash Joins 方式进行连接。

📖 USE_NL

提示 USE_NL 用于指示两个表使用“嵌套循环连接（Nested Loops Join）”方式进行连接（JOIN）。

```
SELECT  /*+ USE_NL(a b) */ a.op_no, a.fnm, b.menu_id, b.gwa_code, b.tr_code,
b.button_flg
FROM tb_user a, tb_user_grant b
WHERE a.op_no = b.op_no;
```

本例中，表 tb_user（别名 a）和表 tb_user_grant（别名 b）使用 Nested Loops Join 方

式进行连接。

🕮 PARALLEL

并行执行提示的语法是：PARALLEL（表名，并行服务器进程的数量）。并行执行使 Oracle 启动多个并行服务器进程共同执行一条 SQL 语句。

```
SELECT  /*+ FULL(a) PARALLEL(a, 4) */ a.op_no, a.fnm
FROM tb_user a;
```

本例中，启动 4 个并行服务器进程，对表 tb_user 执行全表扫描。

🕮 PARALLEL_INDEX

索引并行执行提示的语法是：PARALLEL_INDEX（表名，索引的名字，并行服务器进程的数量）。

```
SELECT /*+ PARALLEL_INDEX (t,kt,3)*/ op_no
FROM tb_user_grant t
WHERE t.menu_id like '027%';
```

本例启动 3 个并行服务器进程来完成索引 kt2 的扫描。



本章总结

本章重点介绍 SQL 语句的优化原理。在本章的实战过程中，你会遇到一些晦涩的概念，这也在情理之中。当我们遇到一些晦涩的概念时，我们可以上网或者查看 Oracle 的官方文档，以获得这些概念的解释，在此过程中，当你明白其中一个概念的时候，你也许也明白了其他引申的概念。在学习数据库原理时，更应该按照这种方法进行学习。

第 18 章　模式对象管理

一个模式（Schema）是一个数据库对象的集合。模式为一个数据库用户所有，其名字与该用户的名称相同。模式对象是由用户创建的逻辑结构，用以包含或引用它们的数据。模式对象包含诸如表（Tables）、索引（Indexes）、视图（Views）、聚簇（Clusters）和同义词（Synonyms）等。以上是模式对象的严格定义，比较难于理解，通俗来说，模式对象就是一个用户，该用户拥有几类对象，我们把这些对象（如表等）叫模式对象。模式对象包括：

- 聚簇（Clusters）
- 约束（Constraints）
- 数据库链接（Database Links）
- 触发器（Database Triggers）
- 维（Dimensions）
- 外部过程库（External Procedure Libraries）
- 索引和索引类型（Indexes And Indextypes）
- Java 类及 Java 资源（Java Classes、Java Resources）
- 物化视图、物化视图日志（Materialized Views、Materialized View Logs）
- 对象表、对象类型、对象视图（Object Tables、Object Types、Object Views）
- 操作符（Operators）
- 序列（Sequences）
- 函数、存储过程和包（Stored Functions、Procedures、Packages）
- 同义词（Synonyms）
- 表、索引组织表（Tables、Index-Organized Tables）
- 视图（Views）

我们不应该花太多时间深刻理解“什么叫模式”、“什么叫模式对象”以及它们的区别，我们只需理解和使用表（Tables）、索引（Indexes）、视图（Views）、聚簇（Clusters）和同义词（Synonyms）等就足够了。说白了，模式名就是用户名，用户名就是模式名！

要想引用自己拥有的对象（如表），只需输入该对象的名字便可；倘若要引用其他用户拥有的对象，需要在对象名前面加上“用户名.”，如用户 kite 要访问用户 itme 的表 tb_user_infor，需要输入：itme.tb_user_infor。



注：有的模式对象已经或者将要在其他章节中进行讲解。



18.1 数据的封藏——视图

视图是表中数据的逻辑表示，视图也可以说成是一段存储的 SELECT 语句，它用于从数据库检索视图要表示的数据。视图并不存放数据，数据真正存放在基表中。同真实的表一样，视图包含一系列带有名称的列和行数据。视图对用户名、基表名、基表数据进行了**封藏**，它只是基表数据展现的“窗口”，而一个基表可以有许多这样的“窗口”（如图 18-1 所示）。视图有如下优点：

- 加强了表的安全管理，用户只能访问视图，不能直接访问表。
- 隐藏了数据的复杂性。
- 简化了 SQL 语句的书写。

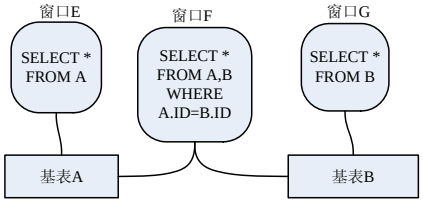


图 18-1　视图的窗口作用



18.1.1 视图创建

创建视图的过程其实很简单，我们应该多花一点时间理解视图的原理。

创建视图的 SQL 语句如下所示：

```
CREATE VIEW v_zero AS --语句1
SELECT * --语句2
FROM kk.students --语句3
WHERE id>1000; --语句4
```

其中，v_zero 是视图的名字；kk 是基表的拥有者；AS 之后指定查询语句；语句 2、语句 3、语句 4 是 SELECT 语句，用于从基表中取数据。SELECT 语句可以是 多表查询，也可以是单表查询，还可以从其他视图中获取数据，也就是在视图上再创建视图。

创建只读视图的 SQL 语句如下所示：

```
CREATE VIEW v_readonly
 AS
SELECT * FROM students
WITH READ ONLY;
```

加上选项 WITH READ ONLY 表示创建只读视图。



知识点索引——只读视图

只读视图，顾名思义，就是只允许被读的视图，不允许对这样的视图执行 DML 操作（UPDATE、INSERT、DELETE）。创建视图时，加上选项 WITH READ ONLY，将创建成只读视图。



18.1.2 在存储过程中创建视图

通常不允许在存储工程中创建视图，但是，通过动态 SQL，允许在存储过程中创建视图。EXECUTE IMMEDIATE 语句用于执行动态 SQL。本文将通过存储过程 p_create_view 创建视图 v_zero3。

创建存储过程 p_create_view。

```
CREATE OR REPLACE PROCEDURE p_create_view
is
sql_string VARCHAR2(500);
BEGIN
sql_string:='CREATE view v_zero3 as SELECT * FROM students'; --创建视图的SQL 语句
EXECUTE IMMEDIATE  sql_string;--执行 SQL 语句，创建视图 v_zero3。
END;
```


执行存储过程 p_create_view。

```
SQL> EXECUTE p_create_view;
PL/SQL procedure successfully completed.
```


查看视图是否创建成功。

```
SQL> DESC  v_zero3
```

可以看出，视图 v_zero3 成功创建。 如果视图创建失败，执行 DESC 命令会提示该视图不存在。



18.1.3 视图的使用

通过学习本章的内容可以帮助我们理解什么是视图。

1．视图查询

通过基表的“窗口（视图）”，我们可以访问基表中的数据。

```
SELECT * FROM v_zero;
```

其中，v_zero 是视图的名字。

就像使用基表查询数据一样，我们可以从视图中查询数据（实质还是从基表中获取数据）。

视图 v_zero 封藏了基表的名字（students）、基表的拥有者（kk）、基表的数据（只能看到 id>1000 的数据）。

2．在视图上执行 DML

通过视图，我们可以对基表的数据进行修改（INSERT、UPDATE、DELETE）。但是，这样的修改有许多限制，并且会给你的开发带来不必要的麻烦，因此，我们**不推荐**通过视图对基表数据进行修改。若要修改数据，请直接通过基表来实现。

通过视图修改基表数据的示例如下：

```
INSERT INTO v_staff
VALUES (1133,'kite');
```

上面的 SQL 语句通过视图 v_staff 往基表中插入数据。



18.1.4 得到创建视图的 SQL 语句

通过 DBMS_METADATA 中的函数 GET_DDL 可以得到创建视图的 SQL 语句，执行过程如下：



设置 SQL*Plus 环境变量。

```
SQL> SET SERVEROUTPUT ON
SQL> SET LONG 100000
```



得到创建视图 V_READONLY 的 SQL 语句。

```
SQL> SELECT DBMS_METADATA.GET_DDL('VIEW','V_READONLY','ITME')
  2  FROM DUAL;
```

SQL 语句的执行结果是：

```
DBMS_METADATA.GET_DDL('VIEW','
------------------------------------------------------------
CREATE OR REPLACE FORCE VIEW "ITME"."V_READONLY" ("NAME", "ID", "DETAIL") AS
SELECT "NAME","ID","DETAIL" FROM students
WITH READ ONLY
```

其中，V_READONLY 是视图的名字，我们要获得创建该视图的 SQL 语句。ITME 是用户名（模式名），是视图 V_READONLY 的拥有者。

需要注意的是，在函数 GET_DDL 中，用户名和视图名必须用大写，否则会报告视图。



18.1.5 视图的删除

删除视图 V_READONLY。

```
SQL> DROP VIEW V_READONLY;
```

删除视图时，如果视图上存在参照完整性约束的引用，则无法删除视图，需要使用选项 CASCADE CONSTRAINTS，表示连参照完成性约束一起删除。



18.2 产生唯一值——序列（Sequence）

在 SQL Server 中，属性 IDENTITY 用于标识自增列，例如创建表 employees。

```
CREATE TABLE employees
(
 ID int IDENTITY(1,1),
 NAME VARCHAR(20)
)
```

表 employees 中的列 ID 将自动增长。

但是，在 Oracle 中，自增字段的实现就复杂些。在 Oracle 中，使用序列实现字段的自增。序列（Sequence）又叫序列生成器，它用于提供一系列的数字，开发人员使用序列生成唯一键。每次访问序列，序列按照一定的规律增加或者减少。序列的定义存储在 SYSTEM 表空间中，序列不像表，它不会占用磁盘空间。序列独立于事务，每次事务的提交和回滚都不会影响序列。



18.2.1 创建序列

创建序列的语法如下所示：

```
CREATE SEQUENCE 序列名字
INCREMENT BY
START WITH 整数
MAXVALUE 整数
NOMAXVALUE
MINVALUE 整数
NOMINVALUE
CYCLE
NOCYCLE
CACHE 整数
NOCACHE
ORDER
NOORDER
```

- INCREMENT BY 指定序列的增量，如果指定负数，序列的值将会出现负增长。
- START WITH 指定序列的起始值。
- MAXVALUE 指定序列的最大值。
- NOMAXVALUE 表示序列无最大值。
- MINVALUE 指定序列的最小值。

- NOMINVALUE 指定序列无最小值。
- CYCLE 指定序列到达最大值时，可以循环。
- NOCYCLE 指定序列到达最大值时不能循环。
- CACHE 指定可以缓存多少个值在内存中，缓存的目的是加快对序列的访问。
- NOCACHE 表示不缓存序列。
- ORDER 表示按照请求的顺序产生序列。
- NOORDER 表示不按照顺序产生序列。

创建序列的例子如下所示：

```
SQL> CREATE SEQUENCE sq2
  2  INCREMENT BY 1
  3  START WITH 1
  4  NOMAXVALUE
  5  NOCYCLE
  6  CACHE 10;
```



18.2.2 序列的使用

NEXTVAL 使序列的值增加。

```
SELECT sq2.NEXTVAL FROM dual;
```

CURRVAL 得到序列的当前值，而不会使序列的值增加。

```
SELECT sq2.CURRVAL FROM dual;
```

序列应用的例子如下所示：



取序列的值。

```
SQL> SELECT sq2.NEXTVAL FROM dual;
   NEXTVAL
----------
         1
```



序列值增加 1。

```
SQL> SELECT sq2.NEXTVAL FROM dual;
   NEXTVAL
----------
         2
```



序列值不变

```
SQL> SELECT sq2.CURRVAL FROM dual;
   CURRVAL
----------
         2
```

STEP 04 再次执行查询，序列值不变。

```
SQL> SELECT sq2.CURRVAL FROM dual;
  CURRVAL
----------
         2
```



注：每个会话执行 CURRVAL 之前，必须执行 NEXTVAL，否则会产生错误“ORA-08002: sequence string.CURRVAL is not yet defined in this session”。



18.2.3 序列值丢失（不连续）的问题

本例中，我们将演示丢失，仍以序列 sq2 为例。

STEP 01 查看序列值，序列值是 2。

```
SQL> SELECT sq2.CURRVAL FROM dual;
  CURRVAL
----------
         2
```

STEP 02 重启数据库。

```
SQL> STARTUP FORCE;
```

该命令先关掉数据库，再重启数据库。

STEP 03 再次查看序列值，序列值部分丢失。

```
SQL> SELECT sq2.NEXTVAL FROM dual;
  NEXTVAL
----------
        11
```

在序列 sq2 的定义中，有一参数 CACHE 10，表示每次缓存 10 个序列值在内存中，这样将加速应用程序对序列的访问。但是，当数据库重启的时候，缓存的序列值将丢失！所以，我们再次查询序列时，看到序列值不连续。在 Oracle 集群环境（Real Application Clusters，RAC）中，也会发生序列的不连续性。定义序列时，最好指定参数 ORDER，让序列值按照请求的先后顺序产生。



18.2.4 把序列的值重新置成 1

目前，Oracle 没有提供把序列值重新置成 1 的命令。若想把序列值重新置成 1，只能先删除原来的序列，然后重建该序列（重建时使用选项 START WITH 1）。



18.2.5 删除序列

删除序列 sq2。

```
SQL> DROP SEQUENCE sq2;
```



18.3 隐藏数据库对象的名字——同义词（SYNONYM）

同义词又叫别名，通俗地说，一个同义词其实就是给一个对象起的别名。别名和对象的名字指向的都是同一模式对象。可以给表、视图、序列、程序、函数或包等模式对象起别名。起别名的目的是让我们很方便地引用模式对象。接下来我们将举例说明。

STEP 01 环境准备。

```
CREATE TABLE chinastudentbook
(a int);

INSERT INTO chinastudentbook VALUES (6);
COMMIT;
```

STEP 02 计算表 chinastudentbook 中的行数。

```
SQL> SELECT COUNT(*) FROM itme.chinastudentbook@dblink6;
  COUNT(*)
----------
         1
```

从这里可以看出，为了引用表 chinastudentbook，我们使用了特别冗长的字符串（itme.chinastudentbook@dblink6）。

STEP 03 创建表 chinastudentbook 的别名 ctb。

```
SQL> CREATE SYNONYM ctb FOR itme.chinastudentbook@dblink6;
```

这里创建的是私有同义词。

其中，CREATE SYNONYM 和 FOR 是关键字。chinastudentbook 是表名。itme 是用户名，该用户拥有表 chinastudentbook。dblink6 是数据库链接的名字，指向远程数据库。

STEP 04 使用别名 ctb 计算表 chinastudentbook 中的行数。

```
SQL> SELECT COUNT(*) FROM ctb;
  COUNT(*)
----------
         1
```

创建表 chinastudentbook 的别名 ctb 后，我们只需引用表的别名即可，引用表的别名和引用表没有什么区别，只是前者更加方便。

18.3.1 同义词的优点

从上面的例子可以看出，同义词有如下优点：

- 隐藏对象的模式名（用户名，如本例的 itme）。
- 提供远程对象的透明性，访问远程对象就好像访问本地对象（隐藏远程数据库链接 dblink6）。
- 简化 SQL 语句。
- 限制用户直接对对象的访问（隐藏对象名）。

18.3.2 同义词的分类

同义词分为公有同义词和私有同义词。

- 公有同义词（Public Synonym），公有同义词被 PUBLIC 用户所拥有，数据库中的任何用户都可以使用公有同义词。
- 私有同义词（Private Synonym），私有同义词被特定的用户所拥有，这个用户控制其他用户对私有同义词的访问。

18.3.3 创建同义词

创建私有同义词的 SQL 语句如下所示：

```
CREATE SYNONYM ctb FOR itme.chinastudentbook@dblink6;
```

其中，dblink6 是同义词的名字；itme 是模式名（用户名）；chinastudentbook 是表名。

创建公有同义词的 SQL 语句如下所示：

```
SQL> CREATE PUBLIC SYNONYM pb_ctb FOR itme.chinastudentbook@dblink6;
```

18.3.4 使用同义词

可以像使用普通对象一样使用同义词，同义词只是实际对象的别名，最终都指向实际存在的对象。

18.3.5 删除同义词

删除私有同义词。

```
SQL> DROP SYNONYM ctb;
```

删除公有同义词。

```
SQL> DROP PUBLIC SYNONYM pb_ctb;
```



本章总结

本章介绍了模式对象的定义，并详细介绍了视图、序列、同义词。在构建视图时，一定要注意视图中 SQL 语句的执行效率，有关 SQL 语句的执行效率，**请参考本书第 17 章的内容。**对于视图，虽然 Oracle 允许你有条件地通过视图更新基表，但是，我们不主张大家这么做！

第 19 章　开发基础的沉淀
——PL/SOL 程序设计

本章将重点介绍如何利用 PL/SQL 开发数据库应用程序。如果你立志做 Oracle 的开发，请你认真阅读本章。本章重在介绍开发，而弱化系统管理。

19.1　PL/SQL 介绍



SQL 已经被发展成一种通用的数据库语言，但是，SQL 只能实现简单的功能，若要完成复杂的任务，SQL 就显得力不从心。于是，Oracle 公司对它进行了扩展，使其功能更加强大，Oracle 把扩展以后的 SQL 叫 PL/SQL。PL/SQL 是一种模块化的程序语言，用于开发基于数据库的应用。PL/SQL 是一种过程化语言，属于第三代语言（3G）。PL/SQL 不是一个独立的产品，它是一种融合到 Oracle 服务器和 Oracle 工具中的技术。

PL/SQL 编译和运行系统是一个 PL/SQL 引擎，其处理 **PL/SQL 单元**，PL/SQL 引擎处理 PL/SQL 程序块。当 PL/SQL 程序块被送到 PL/SQL 引擎处理时，PL/SQL 程序块被分成两部分进行处理：一部分是过程化语句，这部分语句将被 PL/SQL 引擎中的过程化语句执行器（Procedural Statement Executor）所执行；另外一部分语句是 SQL 语句，这部分语句将被发送到 Oracle 服务器的 SQL 语句执行器（SQL Statement Executor）中处理。

PL/SQL 有如下优点：

- 与 SQL 紧密继承，SQL 与 PL/SQL 完美地结合在一起，共同完成数据库开发
- 执行效率高
- 安全
- 可扩展
- 易于管理
- 可访问 Oracle 预定义包
- 支持面向对象技术

PL/SQL 单元包括：块（PL/SQL Block）、函数（Function）、包（Package）、包体（Package Body）、过程（Procedure）、触发器（Trigger）、类型（Type）、类型体（Type Body）。



19.1.1 PL/SQL 基本要素

本节介绍 PL/SQL 程序设计最基本的要素，本节只需读者了解大概即可，不必对每个晦涩的名词作深入理解。学习程序最强调的是应用，不应耗费太多时间停留在语法的细枝末节上。

1．字符单元

编写 PL/SQL 程序时，代码由下面的字符组成：

- 大写字母 A 到 Z、小写字母 a 到 z。
- 数字 0 到 9。
- 符号() + - * / < > = ! ~ ^ ; : . ' @ % , " # $ & _ | { } ? []
- Tab 键、空格键（Spaces）和回车换行（Carriage Return）

几乎任何程序都由这些字符单元组成。在 PL/SQL 中，不区分大小写。

2．词法单元

PL/SQL 词法单元包括分隔符（Delimiters）、标识符（Identifiers）、数值（Literals）和注释（Comments）。

分隔符

分隔符包括+、%、' 、. 、/ 、(、) 、: 、, 、* 、" 、= 、< 、> 、@ 、; 、- 、:=、=>、||、**、<<、>>、/*、*/、..、<>、!=、~=、^=、<=、>=、--，每个分隔符都有特殊的含义。

标识符

标识符用于命名 PL/SQL 单元，这些单元包括常量（Constants）、变量（Variables）、异常 （Exceptions）、游标（Cursors）、游标变量（Cursor Variables）、存储过程（Procedure）、函数（Functiong）和包（Packages）。标识符的最小长度是 1，最大长度是 30。但是，标识符的第一字符必须是字母。

在 Oracle 中，有些保留字（如 END）或者预定义标识符（STANDARD），它们有着特殊的含义。在命名 **PL/SQL 单元**时，不要使用保留字或者预定义标识符。

数值

数值就是一个数字、字符、字符串或布尔（Boolean）值。它本身是数据而不是一个变量。

- 数字值，如 7、 0、 +3、 -14 等。
- 字符值，字符值是单个字符用单引号括起来，如'K' 、'5' 、' ' 、'%'、'z'等。字符值大小写敏感。
- 字符串值，字符串值是把多个字符用单引号括起来，如'Hello,YOU!'、'com'、'**'。

字符串大小写敏感。

- 布尔值，布尔值有三个：TRUE、FALSE、NULL。
- 日期和时间值。

注释

为了增加代码的可读性，可以在程序中增加注释，PL/SQL 会忽略代码中的注释。有两种类型的注释，单行注释和多行注释。

单行注释由一对连字符（--）开头，如：

```
SELECT * --从表中选取所有行的数据
FROM T;
```

多行注释以符号（/*）开头，以符号（*/）结尾，如：

```
DECLARE
/*变量 V_A 定义了……
在使用这个变量时，要…… */
    V_A INT  ;
BEGIN
    NULL;
END ;
```



19.1.2 PL/SQL 块（PL/SQL Block）

PL/SQL 程序是以块（Block）为基本单位，块其实就是一段程序，只是它按照下面的方式进行定义。块可以进行嵌套。

```
[DECLARE]
--declaration statements
BEGIN
--executable statements
[EXCEPTION]
--exception statements
END
```

- DECLARE 声明部分，声明部分以 DECLARE 开始，用于定义变量（常量），包括变量的名字、数据类型、初始值。声明部分是可选的。
- BEGIN 语句执行部分，以 BEGIN 开始，该部分包含了所有的语句和表达式。在该部分可以执行 SQL/SQL 代码和 SQL 语句。语句执行部分是必需的。
- EXCEPTION 异常处理部分，以 EXCEPTION 开始，它也包含一系列代码，用于处理块执行中出现的异常。只有执行部分出现异常时，才会执行到异常处理部分。异常部分是可选的。
- END 标志块的结束。

1．块的嵌套

PL/SQL 块分为匿名块和命名块。一个块中可以嵌套另外一个块（无论其是否命名），并且可以多层嵌套，嵌套的例子如下所示：

```
DECLARE --块 1 的开始
   V_A INT  ;
BEGIN
    NULL;
    BEGIN--块 2 的开始
    NULL;
    END; --块 2 的的结束
EXCEPTION
WHEN OTHERS THEN
    NULL;
END ;--块 1 的的结束
```

如上所示，块 2 被嵌套在块 1 中。

2．匿名块（Anonymous Block）

匿名块其实就是未命名的 **PL/SQL 单元**（尽管块有标签），块被动态生成，块的代码并未存储在数据库中，块可能只是被临时使用。匿名块同样由定义部分、执行部分、异常处理三部分组成。匿名块不能被其他程序所调用。

3．命名块（A Named PL/SQL Block）

命名块是被编译好并存储在数据库中的代码。命名块可以被重复调用。存储过程、函数、包等就是命名块。

4．PL/SQL 子程序（PL/SQL Subprogram）

PL/SQL 子程序就是命名的 PL/SQL 块，我们可以调用子程序，调用时可以指定参数。通常所说的子程序就特指函数和存储过程。



19.1.3 变量和常量

变量（Variable）和常量（Constant）都可以用于存储值，但是，在程序运行期间，变量的值可以不断变化，而常量的值却不可以。

1．变量（常量）的声明

变量声明的格式如下：

```
Variable_name  databyte [NOT NULL][:=|DEFAULT expression]
```

其中，Variable_name 表示变量的名字；databyte 表示变量的数据类型；NOT NULL 表示变量不能为空；DEFAULT 表示指定变量的初始值，也可以不指定变量的初始值。

常量声明的格式如下：

```
Variable_name CONSTANT databyte [NOT NULL][:=|DEFAULT expression]
```

其中，CONSTANT 表示定义的是常量；Variable_name 表示常量的名字；databyte 表示常量的数据类型；NOT NULL 表示常量不能为空；DEFAULT 指定常量的值。

例如：定义常量 c_id。

```
c_id CONSTANT NUMBER := 50;
```

2．变量赋值

通过赋值操作符**:=**给变量赋值。如：给变量 v1 赋予值 France。

```
v1 := 'France';
```

赋予布尔值。

```
done := FALSE;
```

3．变量的作用范围

变量的作用范围，指一个变量在块的哪些部分可以被引用。在一个块（主块）中定义的变量，在该块的子块中也有效。在一个块中不能出现相同的变量定义，但是在主块和子块中可以出现相同的变量定义。

■ 全局变量的例子——在子块内也可以引用全局变量

```
SQL> DECLARE --块 1 的开始
  2    m VARCHAR2(20);
  3  BEGIN
  4    m := '我是大 M';
  5
  6    DECLARE --块 2 的开始
  7    BEGIN
  8      dbms_output.put_line(m);--在子块内引用全局变量
 10    END; --块 2 的开始
 11
 12      dbms_output.put_line(m);--在子块外引用全局变量
 13  END;--块 1 的结束
 14  /
我是大 M
我是大 M
```

这段代码的输出是“我是大 M 我是大 M”。

块 1 是主块，块 2 是块 1 的子块。在块 1 中定义变量 m，该变量在所有的块中都有效（包括在子块中），在子块中可以引用全局变量 m，在子块外也可以引用全局变量 m。

■ 局部变量使用的例子

```
SQL> DECLARE --块 1 的开始
  2    m VARCHAR2(20); --定义大 M
  3  BEGIN
  4    m := '我是大 M';
  5
  6    DECLARE --块 2 的开始
  7      m VARCHAR2(20);--定义小 m，只作用在子块内
```

```
 8   BEGIN
 9     m := '我是小 m';
10     dbms_output.put_line(m);
12   END; --块 2 的结束
13
14    dbms_output.put_line(m);
15  END;--块 1 的结束
16  /
我是小 m
我是大 M
```

这段代码的输出是“我是小 m 我是大 M”。在主块 1 中定义了全局变量 m，在字块 2 中定义了一个局部变量 m，这两个变量并不冲突（虽然名字相同），块 1 中定义的 m 是全局变量，作用于全局；块 2 中定义的 m 是局部变量，只作用于块 2 内部。

■ 全局变量和局部变量名字相同的对策

如果在子块中同时需要使用相同名字的全局变量和局部变量，需要在全部变量前指定标签。但是，如果出现这种情况，我们强烈推荐使用不同的变量名，使用不同的变量名可以增加程序的可读性，减少程序调试的成本。

4．绑定变量（Bind Variables）

绑定变量是在 SQL*Plus 创建和使用的变量，创建绑定变量的目的是：重复利用 SQL 语句。绑定变量可以被 SQL*Plus 识别。绑定变量可以在 PL/SQL 和 SQL 中使用。绑定变量使用的例子如下：

STEP 01 创建绑定变量 bd。

```
VARIABLE bd VARCHAR2(100);
```

其中，VARCHAR2(100)用于定义绑定变量的数据类型。

特别需要注意的是，VARIABLE 是 SQL*Plus 命令，用于定义绑定变量。SQL*Plus 命令不是 SQL 语句，也不是 PL/SQL 代码，SQL*Plus 命令脱离 SQL*Plus 将不能运行（我们在前面已经做了讲解，**请参考本书 5.1 节的内容**）。

STEP 02 在 PL/SQL 块中引用绑定变量。绑定变量的引用格式是冒号（:），后面紧跟绑定变量的名称（注意“:=”是赋值操作符）。

```
SQL> BEGIN
2  :bd:='hello YOU!';
3  END;
```

STEP 03 打印绑定变量的内容。

```
SQL> PRINT bd;
BD
--------------------------------------------
Hello YOU!
```

同理，PRINT 也是 SQL*Plus 命令。



19.1.4 数据类型

PL/SQL 数据类型和数据库数据类型有些差异，有的数据类型相同，有的数据类型不同。PL/SQL 的数据类型分成两大类：

- 预定义 PL/SQL 数据类型（Predefined PL/SQL Data Types）
- 用户自定义 PL/SQL 子类型（User-Defined PL/SQL Subtypes）

1．预定义 PL/SQL 数据类型

PL/SQL 预定义数据类型包括：

- 标量类型（Scalar，就是标准的数据类型），标量类型没有内部组件。
- 复合类型（Composite），复合类型包含了能够被单独操作的内部组件。
- 引用类型（Reference），引用类型类似于 3G 语言中的指针。
- LOB 类型（Large Object），LOB 类型的值就是一个 Lob 定位器。

上述四大类的细分如表 19-1 所示。

表 19-1　数据类型

标量类型（Scalar）	数字型	BINARY_INTEGER	存储有符号整数，范围从-2,147,483,648 到 2,147,483,648
		PLS_INTEGER	与 BINARY_INTEGER 基本相同，但采用硬件运算，因此，PLS_INTEGER 的运算速度更快
		BINARY_FLOAT	单精度浮点数，用作浮点运算
		BINARY_DOUBLE	双精度浮点数，用作浮点运算
		NUMBER	存储整数或者小数，取值范围从 1E-130 到 1.0E126
	字符型	CHAR	定长字符串，最大长度为 32767 字节
		VARCHAR2	变长字符串，最大长度为 32767 字节
		RAW	存储二进制数据或字节串，最大长度是 32767 字节。RAW 比较像 VARCHAR2 类型（除了 PL/SQL 不解析它之外）
		NCHAR	存储定长国家字符集数据
		NVARCHAR2	存储变长国家字符集数据
		LONG	存储二进制数据或二进制字符串，最大长度是 32760 字节。这种数据类型已经逐渐被 LOB 替换，建议不要使用
		LONG RAW	存储变长的字符串，最大长度是 32760 字节。这种数据类型已经逐渐被 LOB 替换，建议不要使用

续表

		ROWID	存储行的逻辑地址
		UROWID	存储行的物理地址
	布尔型	BOOLEAN	存储逻辑值：真（TRUE）、假（FALSE）
	日期型	DATE	日期时间
		TIMESTAMP	时间戳，它是 DATE 的扩展，更加精确（DATA 只精确到秒）
		TIMESTAMP WITH TIME ZONE	有时区的时间戳
		TIMESTAMP WITH LOCAL TIME ZONE	有本地时区的时间戳
		INTERVAL YEAR TO MONTH	存储时间间隔，其单位是年和月，如：1 年零 3 个月
		INTERVAL DAY TO SECOND	存储时间间隔，其单位是天和秒，如：4 天零 5 小时 5 分 20 秒
集合类型（Composite）		NESTED TABLE	嵌套表
		VARRAY	可变数组（变长数组）
		TABLE	表（这里不是普通的表，是一种数据类型）
		RECORD	记录
指针类型（Reference）		REF CURSOR	游标类型
大对象类型（Large Object）		BFILE	存储二进制对象（指针），实际内容存放到操作系统的文件中
		BLOB	存储大的二进制数据，最大存储 128TB
		CLOB	存储大的字符数据，最大存储 128TB
		NCLOB	存储大的国家字符集数据，最大存储 128TB

2．用户自定义 PL/SQL 子类型

一个用户自定义 PL/SQL 子类型其实就是其他数据类型的子集。自定义 PL/SQL 子类型的格式如下所示：

```
SUBTYPE 子类型的名字 IS 基本数据类型[(constraint)] [NOT NULL];
```

例：

```
DECLARE
    SUBTYPE sub_num IS NUMBER(5,1);
    v2 sub_num ;
BEGIN
    NULL;
```

```
END;
```

其中，sub_num 就是 NUMBER 的一个子类型，并且成为了一种新的数据类型，使用这种新的数据类型，我们可以定义变量 v2。

3．CHAR、VARCHAR、VARCHAR2 的区别

对于初学Oracle开发的朋友来说，经常混淆数据类型CHAR、VARCHAR、VARCHAR2，因为这三种数据类型有太多相似之处，它们都用于存放字符数据。下面，我们将告诉大家这三者的区别。

CHAR 是固定长度的，而 VARCHAR2 的长度是可变的，例如，用 CHAR (10)、VARCHAR2 (10)同时存储字符串'ccc'，对于 CHAR (10)，存储的字符将占 10 个字节；而对于 VARCHAR2 (10)，则只占用 3 个字节。

CHAR 的效率比 VARCHAR2 的效率略高。但是，CHAR 在程序开发中的可维护性不高。CHAR 和 VARCHAR2 互为补充，CHAR 浪费空间，但是效率比较高；VARCHAR2 节省空间，但是效率不如 CHAR。

VARCHAR 是 VARCHAR2 的子类型，VARCHAR 可能在以后会变成一种独立的数据类型。出于兼容性的考虑，Oracle 建议使用 VARCHAR2 而不是 VARCHAR（实际开发中也是这样）。

三种数据类型（CHAR、VARCHAR、VARCHAR2）的选择，从兼容性的角度考虑，建议使用 CHAR 和 VARCHAR2（不要使用 VARCHAR）；从节省空间的角度考虑，建议使用 VARCHAR2；从性能的角度考虑，建议使用 CHAR。综合考虑，性价比最高的还是 VARCHAR2。

4．空值（NULL）的问题

NULL 是一个值。空值（NULL）表示未知的意思。空值（NULL）还可以出现在 PL/SQL 中，表示空语句（但只是一条 PL/SQL 语句，这条语句什么都不做）。利用空值（NULL）的例子如下所示：

```
DECLARE
   v2 VARCHAR2(20);
BEGIN
    v2:=NULL;--把空值赋予变量 v2
    NULL; --执行一条空语句，这里是一条语句，不是一个值。
    IF V2 IS NULL THEN dbms_output.put_line('NULL');--判断变量 v2 是否为空
    END IF;
    INSERT INTO kk VALUES(NULL);--往表 kk 中插入空控制。
    COMMIT;
END ;
```

5．数据类型转换

有时，需要把一个值从一种数据类型转换成另外一种数据类型，PL/SQL 支持隐式数

据类型转换和显式数据类型转换。

- 隐式数据类型转换由 PL/SQL 在不知不觉中自动完成。如字符“111”到数字 111 的转换。隐式数据类型转换存在不可预料的转换结果，因此我们推荐使用显示数据类型转换。
- 显示数据类型转换需要开发人员的参与，需要开发人员利用函数完成数据类型的转换。如将日期型转换成字符型，to_char(SYSDATE, 'YYYY-MM-DD:HH24:MI:SS')，其中 SYSDATE 表示系统时间。



19.1.5 PL/SQL 表达式和运算符

表达式由值、变量、运算符和函数组成，PL/SQL 按照一定的规则对其进行处理，并最终产生一个值。PL/SQL 的运算符分成 4 大类：算术运算符、逻辑运算符、比较运算符、连接运算符，如表 19-2 至表 19-6 所示。

表 19-2 算术运算符

运 算 符	含 义
+	加
–	减
•	乘
/	除
**	乘方

表 19-3 逻辑运算符

运 算 符	含 义
AND	两者同时为真结果才为真（TRUE）
OR	只要其中一个为真，结果就为真（TRUE）
NOT	取反

表 19-4 比较运算符

运 算 符	含 义
IS NULL	如果测试值为 NULL，则返回 TRUE；如果测试值不为 NULL，则返回 FALSE
LIKE	比较字符串值
BETWEEN	测试值是否在指定的范围之内
IN	测试操作数是否在一系列的值中

表 19-5 关系运算符

运 算 符	含 义
=	等于

续表

运 算 符	含　义
<>、!=、~=、^=	不等于
<	小于
>	大于
<=	小于或等于
>=	大于或等于

表 19-6 连接运算符

运 算 符	含　义
\|\|	连接符用于连接两个字符串



19.1.6　PL/SQL 语句

PL/SQL 块中的每一条语句都必须以分号结束，一条 PL/SQL 语句可以跨越多行，但要以分号结束，分号表示该语句的结束。一行中可以有多条 PL/SQL 语句，它们之间以分号分隔。**表达式**不是 PL/SQL 语句，但表达式可以是 PL/SQL 语句的一部分。如 5*2 是一个**表达式**，但不是一条 PL/SQL 语句，5*2 只能作为 PL/SQL 语句的一部分，如 a:=5*2;。



19.1.7　在 PL/SQL 中使用 SELECT 语句

在 PL/SQL 中使用查询，必须把查询结果赋予变量。使用 INTO 子句把查询结果赋予变量，INTO 后面指定变量的名字。

```
SELECT ID,NAME
INTO v_id,v_name
FROM TB_ORDERNARY
```

把查询结果值赋予变量，列 ID 的值被赋予变量 v_id，列 NAME 的值被赋予变量 v_name。



19.1.8　在 PL/SQL 中使用 DML

在 PL/SQL 中可以直接使用 DML 语句（INSERT、UPDATE、DELETE）和事务控制语句（COMMIT、ROLLBACK 等）。

```
DECLARE
x VARCHAR2(20) := 'xxxx';
y VARCHAR2(25) := 'yyyyyyyy';
BEGIN
INSERT INTO tb_dmluse VALUES(x, y);--往表中插入数据
```

```
UPDATE tb_dmluse SET name = x WHERE adrress = y;--更新表中的数据
DELETE FROM tb_dmluse WHERE name = x;--从表中删除数据
COMMIT;
END;
```

19.1.9 控制结构

PL/SQL 提供的控制结构用于控制 PL/SQL 语句的执行，有 3 种控制结构：顺序结构、条件结构、循环结构，如图 19-1 所示。

- 顺序结构：按照语句的出现顺序逐条执行。
- 条件结构：只有满足某些条件，语句才被执行。
- 循环结构：语句被循环执行。

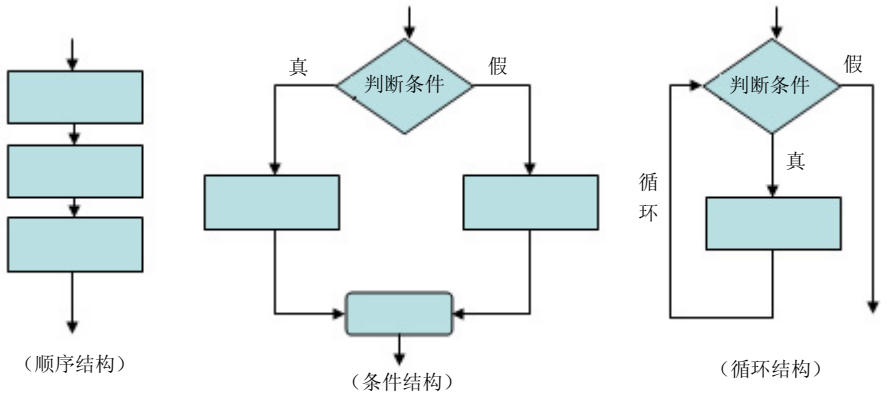


图 19-1　PL/SQL 语句的结构

1．条件结构

条件结构需要对条件进行测试，如果测试结果为真（True），则执行一部分代码；如果测试结果为假（False），则执行另外一部分代码。条件结构包括两个语句：IF 语句和 CASE 语句。

IF 语句的结构如下所示：

```
IF 条件1  THEN
语句1;
语句2;
....
ELSIF 条件2
语句3;
语句4;
....
ELSE
语句5;
语句6;
```

```
....
END IF
```

如果满足“条件 1”，则执行“语句 1”和“语句 2”；如果“条件 1”不满足，则对“条件 2”进行判断，如果满足“条件 2”，则执行“语句 3”和“语句 4”；如果什么条件都不满足，则执行“语句 5”和“语句 6”。

CASE 的用法，**请参考本书 16.5.3 节的内容**。

2．循环结构

循环结构用于重复执行指定的 PL/SQL 语句。有三种类型的循环：基本循环、FOR 循环、WHILE 循环。

基本循环

基本循环的结构如下所示：

```
LOOP
语句1;
语句2;
....
END LOOP
```

Oracle 会一直循环执行 LOOP 到 END LOOP 之间的语句。

FOR 循环

FOR 循环的结构如下所示：

```
FOR counter IN [REVERSE] 开始值..结束值 LOOP
语句1;
语句2;
....
END LOOP
```

其中，“开始值”表示循环的起始值；“结束值”表示循环结束值。Oracle 从“开始值”开始循环，执行语句 1 与语句 2，每循环一次，counter 的值就会加 1，直到 counter 的值等于“结束值”，循环才结束。

在循环的计数方式中，通常都是从“开始值”到“结束值”（即从低到高）；如果指定关键字 REVERSE，表示反向循环，循环计数从“结束值”到“开始值”（即从高到低）。

WHILE 循环

WHILE 循环的结构如下所示：

```
WHILE "条件" LOOP
语句1;
语句2;
....
END LOOP
```

当 Oracle 执行到循环语句时，会对“条件”进行判断，如果条件为真（True），将执行循环内的语句；当循环执行完成后，Oracle 继续对“条件”进行判断，如果满足条件，将继续执行循环内的 SQL 语句。如果不满足“条件”，将中止循环的执行。

循环的退出（EXIT）和继续（CONTINUE）

循环退出

在循环执行进行中，我们可以退出循环（所谓“退出循环”，即停止循环）。有两种方式可以推出循环：第一种方式叫无条件退出（EXIT）；第二种方式叫有条件退出（EXIT-WHEN）。

■ 无条件退出

```
LOOP
...
EXIT;
...
END LOOP;
```

当执行到语句 EXIT 时，循环被无条件终止，跳出到 END LOOP 之后，执行 END LOOP 之后的其他代码。

■ 有条件退出

```
LOOP
...
EXIT WHEN y > 3;
...
END LOOP;
```

当执行到语句 EXIT 时，要对表达式（y>3）进行测试，如果测试结果为真（True），将退出循环，跳到 END LOOP 之后，执行其他代码；如果测试结果为假（False），将继续循环。

循环继续

循环继续同样也有两种方式：无条件继续（CONTINUE）和有条件继续（CONTINUE-WHEN）。

■ 无条件继续

```
LOOP
...
CONTINUE;
...
END LOOP;
```

当执行到 CONTINUE 语句时，将无条件终止本次循环的执行（即不执行 CONTINUE 后的代码），继续执行下一次循环。

■ 有条件继续

```
LOOP
...
CONTINUE WHEN y > 3;
...
END LOOP;
```

当执行到 CONTINUE WHEN 语句时，将对表达式进行测试，如果测试结果为真（True），将终止本次循环，开始下一次循环；如果测试结果为假（False），将继续执行 CONTINUE WHEN 之后的代码。



19.1.10 GOTO 语句

GOTO 语句可以让我们很自由地把程序的执行方向指向另外的代码。在使用 GOTO 语句执行跳转之前，必须指定标签（Label），所谓指定标签，其实就是指定程序的一个断点，跳转以后，程序将从断点后的代码开始执行。使用<<>>来定义标签。需要注意的是，GOTO 有诸多限制，使用时一定要谨慎。GOTO 使用示例如下所示：

```
DECLARE
    …
BEGIN
    <<lb2>>--定义标签
    …
    IF V1 > 20 THEN
    GOTO lb2;--执行跳转
    END IF;
    …
END;
```

本例中，当程序执行到 GOTO 语句时，不再执行 GOTO 之后的语句，而是跳转到标签 lb2 处，从 lb2 后的代码开始执行。



19.1.11 异常（Exception）

在 PL/SQL 中，一个错误条件就叫一个异常（Exception）。在 Oracle 中，有 Oracle 内部定义的异常（如 ORA-01219，其中 01219 是错误编号），也有用户自定义的异常。有的异常还有自己的名字（见表 19-7，如 ZERO_DIVIDE）。当一个错误发生时，一个异常被触发。在 PL/SQL 中，有特殊的机制来处理异常。



注：“异常”是 Oracle 的官方定义，听起来比较别扭，也比较难于理解，通俗地说，“异常”就是错误。

表 19-7　Oracle 预定义异常

异常名字	错误代码	产生原因
NO_DATA_FOUND	01403	SELECT INTO 未返回行，或者你的程序引用了嵌套表中一个已经删除的元素，或者引用表（类型）中一个未初始化的元素
TOO_MANY_ROWS	01422	执行 SELECT INTO 时，返回结果超过一行
ZERO_DIVIDE	01476	除数为 0
ACCESS_INTO_NULL	06530	为一个未初始化的对象属性赋值
CASE_NOT_FOUND	06592	在 CASE 过程中 WHEN 后没有包含必要的条件分支并且没有 ELSE 子句
COLLECTION_IS_NULL	06531	集合元素未初始化
CURSER_ALREADY_OPEN	06511	游标已经打开
DUP_VAL_ON_INDEX	00001	重复值
INVALID_CURSOR	01001	在不合法的游标上进行操作
INVALID_NUMBER	01722	字符转换为数字错误
SUBSCRIPT_BEYOND_COUNT	06533	元素下标超过嵌套表或 VARRAY 的最大值
SUBSCRIPT_OUTSIDE_LIMIT	06532	使用嵌套表或 VARRAY 时，将下标指定为负数
VALUE_ERROR	06502	赋值时，数据长度超出目标字段定义的长度

大家还记得吗？在定义 PL/SQL 块的时候，有一部分叫“异常处理部分”，“异常处理部分”用于对异常进行处理。块中异常处理部分的定义如下所示：

```
DECLARE
 ...
BEGIN
 ...
EXCEPTION
 WHEN 异常 1 THEN
 对异常 1 进行处理;
 WHEN 异常 2 THEN
 对异常 2 进行处理;
 ...
 WHEN OTHERS THEN
 ...
END;
```

其中，“EXCEPTION”表示异常处理的开始。“WHEN 异常 1 THEN”表示当出现“异常 1”时，如何对“异常 1”进行处理。“WHEN OTHERS THEN”表示如果出现的是其他异常，将如何进行处理。

1．捕捉异常

在异常处理部分，如果我们确实不知道出现的异常（错误），也不知道该如何处理，我们可以在此捕获错误代码和错误信息，然后根据错误信息定位错误的原因。捕获异常的

范例如下所示：

```
BEGIN
……
EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE('Error code:' ||SQLCODE||'        '||SQLERRM);
END;
```

本段代码将捕获块中出现的错误（包括错误代码和错误信息）。SQLCODE 和 SQLERRM 都是 Oracle 预定义函数，SQLCODE 返回错误代码，SQLERRM 返回错误信息。本例中，DBMS_OUTPUT.PUT_LINE 将打印错误代码和错误信息。

2．定义你的异常

在上面我们说过，除了 Oracle 预定义异常，我们也可以定义自己的异常。

- Oracle 预定义的异常：

ORA-1034 ORACLE not available

- 自定义的异常：

ORA-20002 my value is too low
错误编号和错误信息都是我们定制的。

自定义异常的格式

异常的定义格式如下所示：

```
“异常的名字” EXCEPTION;
```

这里定义了异常的名字。

给异常指定错误编号

当然，我们也可以给异常指定错误编号。

```
PRAGMA EXCEPTION_INIT(exception_name, -Oracle_error_number);
```

其中，exception_name 是异常的名字；Oracle_error_number 是错误编号，以符号（-）开始，范围从-20000 到-20999。

异常触发的方式

有三种方式触发异常：

- 由 Oracle 自动触发异常。
- 使用 RAISE 语句手工触发。
- 调用存储过程 RAISE_APPLICATION_ERROR 手工触发。

示例 1　使用 RAISE 触发。

```
DECLARE
  myecp EXCEPTION;--语句 1
```

```
BEGIN
--…
RAISE myecp;--语句 2
--…
EXCEPTION WHEN myecp THEN--语句 3
--…
dbms_output.put_line('myecp error.......');
--…
END;
```

其中，“语句 1”定义了异常 myecp。“语句 2”触发异常 myecp。“语句 3”对异常 myecp 进行处理。

需要注意的是，RAISE 也可以触发数据库预定义异常。



示例 2　使用 RAISE_APPLICATION_ERROR 触发。

```
DECLARE
  myecp2 EXCEPTION;--语句 1
  PRAGMA EXCEPTION_INIT(myecp2, -20009); --语句 2
BEGIN
  --…
  RAISE_APPLICATION_ERROR(-20009, 'data is out of list');--语句 3
  --…
EXCEPTION WHEN myecp2 THEN--语句 4
  --…
  DBMS_OUTPUT.PUT_LINE('Error code:' ||SQLCODE||'        '||SQLERRM); --
语句 5
  --…
END;
```

其中，“语句 1”定义异常 myecp2；“语句 2”给异常 myecp2 一个错误编号 20009（注意负号）。EXCEPTION_INIT 过程用于把一个错误编号赋予一个异常；“语句 3”触发异常。触发异常时，还指定了错误信息（'data is out of list'）；“语句 4”对异常进行处理；“语句 5”输出异常的错误代码和错误信息。

3．异常的传递

在一个块中，只能有一个异常处理部分，在不同的块中，可以定义多个异常处理部分。父块中可以定义异常处理，子块中也可以定义自己的异常处理。当一个子块的异常发生时，如果在子块的异常处理部分没有进行处理，该异常会传递到父块中，如果父块没有异常处理部分，该异常会传递到父块的父块中。

4．处理异常

在异常处理部分，需要开发人员编写代码，针对不同的异常进行不同的处理。需要注意的是，如果块的处理部分已经有 DML 语句对数据库进行了修改，我们要慎重考虑，是要在异常处理部分提交（COMMIT），还是回滚（ROLLBACK）前面的 DML 语句对数据库的修改。

19.2 数据批处理——游标（Cursor）

PL/SQL 中，我们可以使用 SELECT INTO 语句获得一行数据，并对其进行处理。可是，倘若我们要对多行数据进行处理，该怎么办呢？不要担心，游标正好帮我们实现处理多行数据的功能。一个游标是一个指定的私有 SQL 区（Private SQL Area）的句柄，或者是一个命名的私有 SQL 区，这片区域存放解析过的语句和其他信息。

19.2.1 游标的分类

Oracle 中的游标有两类：显式游标、隐式游标。

1．隐式游标（Implicit Cursor）

隐式游标又叫 SQL 游标（SQL Cursors），是在执行插入 DML 操作（INSERT、DELETE、UPDATE）和 SELECT 语句返回单条记录时由 PL/SQL **自动、隐含定义**。隐式游标由 PL/SQL 自动定义、自动打开、自动关闭、不需要用户的参与。隐式游标的游标名叫 SQL。

游标属性用于返回DML和DDL执行的信息。隐式游标有4大属性：%FOUND、%NOTFOUND、%ISOPEN、%ROWCOUNT。其中%FOUND、%NOTFOUND、%ISOPEN的结果为BOOLEAN型，%ROWCOUNT的结果为整数。通过SQL%的形式引用隐式游标的属性。

- %FOUND，用于判断 DML 语句是否改变了行（或者判断 SELECT INTO 是否返回了一行或者多行数据）。
- %NOTFOUND，意思正好和 SQL%FOUND 相反。
- %ISOPEN，判断游标是否打开，对隐式游标来说，当 SQL 语句执行完成后，游标被自动关闭，因此 SQL%ISOPEN 的值永远是 False。
- %ROWCOUNT，用于判断 DML 语句影响了多少行，或者 SELECT INTO 返回了多少行。

示例 1 %FOUND 应用。

```
BEGIN
DELETE FROM TRA;
IF SQL%FOUND THEN --如果删除成功，进行提交
  COMMIT;
ELSE
   NULL;
END IF;
END;
```

从表 TRA 中删除数据，通过 SQL%FOUND 进行判断（SQL 表示隐式游标），如果删除成功，则要进行提交。



示例 2 %ROWCOUNT 应用。

```
SQL> BEGIN
  2  DELETE FROM TRA;
  3  DBMS_OUTPUT.PUT_LINE(TO_CHAR(SQL%ROWCOUNT));
  4  END;
  5  /
2
PL/SQL procedure successfully completed
```

DBMS_OUTPUT.PUT_LINE 输出表 TRA 中有多少行数据被删除。

2．显示游标（Explicit Cursors）

显示游标用 CURSOR...IS 命令定义，它可以对查询语句（SELECT）返回的多条记录进行处理。显示游标需要用户显示进行定义、显示打开和显示关闭。

■ 定义游标（Declaring a Cursor）

声明游标的语法如下所示：

```
CURSOR cursor_name IS SELECT_statement;
```

其中，cursor_name 是游标的名字；SELECT_statement 是查询语句。

如：

```
DELCARE
CURSOR ct IS SELECT *
FROM ttc ;
BEGIN
```

其中，ct 是游标的名字。

■ 打开游标（Opening a Cursor）

打开游标的语法如下所示：

```
OPEN cursor_name
```

使用游标中的值之前，必须首先打开游标，使用 OPEN 打开游标，其中，cursor_name 是在声明部分定义的游标的名字。

如：

```
OPEN CT;
```

■ 关闭游标（Closing a Cursor）

关闭游标的语法如下所示：

```
CLOSE cursor_name
```

如：

```
CLOSE CT;
```

使用完游标之后，要记得关闭游标，未关闭的游标占用内存，耗费系统资源。如果数据库中存在很多未关闭的游标，还可能导致数据库死机。

■ 从游标中取数据（Fetching with a Cursor）

从游标中提取数据的语法如下所示：

```
FETCH cursor_name INTO variable[,variable,...]
```

使用 FETCH 命令从游标中提取数据，每提取一次，游标都指向结果集的下一行。其中，variable 是变量，从游标中取出的数据就放在变量中。

隐式游标有 4 大属性，显示游标也有 4 属性，只是它们的意思不尽相同，需要你区别对待。

- %FOUND 为布尔型属性，当最后一次读（FETCH）记录成功，则返回值为 TRUE。
- %NOFOUND 为布尔型属性，与%FOUND 的意思正好相反。
- %ISOPEN 为布尔型属性，判断游标是否打开，当游标已打开时返回 TRUE。
- %ROWCOUNT 为数字型属性，返回已从游标中读取的记录数（到目前为止）。

示例：显示游标属性。

下面的代码将从奖金表 t_bonus 中取出奖金小于 500（bonus<500）的员工编号（employee_id），并做相应处理。

```
DECLARE
CURSOR c1 IS SELECT EMPLOYEE_ID FROM T_BONUS WHERE BONUS<500;--语句 1
V_EMPLOYEE_ID VARCHAR2(20);
BEGIN
  OPEN c1;--语句 2
  LOOP
    FETCH c1 --语句 3
      INTO V_EMPLOYEE_ID;
    EXIT WHEN c1%NOTFOUND OR c1%NOTFOUND IS NULL;--语句 4
    --...
END LOOP;
--...
CLOSE c1;--语句 5
END;
```


其中，“语句 1”定义游标 c1。“语句 2”打开游标 c1。“语句 3”从游标 c1 提取数据，并放到变量 v_employee_id 中。“语句 4”对游标 c1 的属性 NOTFOUND 和 NOTFOUND 进行判断，如果为真，将退出游标循环，否则继续循环取游标中的数据。“语句 5”关闭游标。



19.2.2 游标 FOR 循环

在使用游标的时候，通常我们都会经历下面的过程：

STEP 01 打开游标。

STEP 02 开始循环。

STEP 03 从游标中取值。

STEP 04 处理从游标中取出的数据。

STEP 05 退出循环。

STEP 06 关闭游标。

上面的过程比较烦琐，因此 Oracle 公司引入了 FOR 循环。FOR 循环把上面的过程融合在一起，只是按照正常的声明方式声明游标。它的优点在于，忽略了显式地打开游标、关闭游标、从游标中取数据等的过程。使用隐式游标，系统隐含定义了一个%ROWTYPE 类型的记录。游标 FOR 循环的语法如下所示：

```
FOR record_name IN
(corsor_name)| (query_difinition)
LOOP
statements
END LOOP;
```

其中，record_name 指定记录的名字；corsor_name 指定游标的名字；query_difinition 表示如果不指定游标的名字，可以指定一个查询；statements 表示这些语句将对游标进行处理；LOOP 是游标循环的开始；END LOOP 是游标循环的结束。

示例：游标 FOR 循环。本例将从游标 c1 中取数据，然后在屏幕上打印出来。

```
DECLARE
CURSOR c1 IS SELECT EMPLOYEE_ID FROM T_BONUS WHERE BONUS<500;--语句 1
BEGIN
--...
FOR item IN c1                                          --语句 2
LOOP    --游标 FOR 循环开始
DBMS_OUTPUT.PUT_LINE(item.EMPLOYEE_ID);                     --语句 3
END LOOP;   --游标 FOR 循环结束
--...
END;
```

其中，“语句 1”定义游标 c1；“语句 2”是游标 c1 的 FOR 循环的开始，item 是记录类型，不需要在块的定义部分进行定义，可以为任意名字。“语句 3”输出记录的内容（员工编号）。



19.2.3 向游标中传递参数

利用游标参数，可以向游标中传递值，这将动态改变游标返回的结果集。我们将利用一个完整的例子，向各位讲解游标参数的用法。

数据库中有一个学生表 students，其信息如下所示：

```
SQL> SELECT * FROM students;

        ID   NAME    CITY
----------  --------    ----------------------------------------
        22   Kite        london
        33   Mary    beijing
        32   TOM     Beijing
```

STEP 01 创建存储过程 p_cur。

存储过程根据学生所在的城市返回学生的信息。

```
CREATE OR REPLACE PROCEDURE p_cur(p_city1 VARCHAR2 )-- 参数 p_city1 表示学生所在城市。
AS
  v_ID      students.id%TYPE;
  v_Namestudents.name%TYPE;
  v_Citystudents.city%TYPE;

  CURSOR c_s(p_ccity2 VARCHAR2) IS -- p_ccity2 是游标的参数，表示学生所在城市
    SELECT id, name, city
      FROM students
      WHERE city = p_ccity2;--游标的内容会根据传入参数值的不同动态变化
BEGIN

  OPEN c_s(p_city1);--用存储过程的传入参数 p_city1，打开游标
  LOOP
    FETCH c_s INTO v_ID,v_Name,v_City;--从游标中取数据
    EXIT WHEN c_s%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_ID||'   '||v_Name||'    '||v_City);--打印游标内容
  END LOOP;

  CLOSE c_s;--关闭游标
END;
```

STEP 02 打开 SERVEROUTPUT。

```
SQL> SET SERVEROUTPUT ON;
```

如果不打开SERVEROUTPUT，屏幕将不会输出DBMS_OUTPUT.PUT_LINE打印的内容。

STEP 03 执行存储过程。

```
SQL> EXEC p_cur('beijing');
33  Mary        beijing
32  TOM     Beijing
```

从这里可以看出，存储过程输出城市是北京（beijing）的学生的信息。



19.2.4 游标变量

游标变量其实就是指针，只是这个指针可以指向不同的查询工作区（Query Work Area），而普通游标（显示游标）总是指向相同的查询工作区（Query Work Area）。简单地说，游标变量是一个变量，只是这个变量可以动态指向不同的游标（区域），因此，我们把游标变量称为**动态游标**，与之相反，我们把显示游标称为**静态游标**。游标变量可以作为函数或者存储过程的参数。

1．定义游标变量

定义游标变量需要两个过程：先定义 CURSOR 类型的指针，再声明游标变量。

STEP 01 定义 CURSOR 类型的指针。

在创建游标变量之前，我们必须定义 REF CURSOR 类型。定义 REF CURSOR 类型的语法如下：

```
TYPE ref_type_name IS REF CURSOR [RETURN return_type]
```

其中，ref_type_name 是新类型的名字，是游标变量中使用的类型，这里实际上定义的是一种新的数据类型。return_type 是可选的，用于指定游标变量返回值的类型，它必须是一个记录类型（RECORD）或行类型（ROWTYPE），如果有返回类型，我们称为“强 REF CURSOR”；如果没有返回类型，我们称为“弱 REF CURSOR”。如果使用“弱 REF CURSOR”，则允许把游标变量与任何查询进行关联；如果使用“强 REF CURSOR”，则只允许把游标变量与特定查询进行关联。

示例 1 定义强 REF CURSOR。

```
TYPE strong_type IS REF CURSOR RETURN employees%ROWTYPE;
```

有返回类型。

示例 2 定义弱 REF CURSOR。

```
TYPE weak_type IS REF CURSOR;
```

没有返回类型。

STEP 02 声明游标变量。

声明游标变量的格式如下所示：

```
strong_type v2
```

其中，strong_type 是上面定义的数据类型；v2 是游标变量的名字。

2．游标变量的使用

接下来，我们将通过一个存储过程（p_cursor）演示游标变量的使用。

```
CREATE OR REPLACE PROCEDURE p_cursor(choice INT) AS
  TYPE newcursor_type IS REF CURSOR;--定义游标变量之前，要定义一种新数据类型（REF CURSOR）
  v_cursor newcursor_type;--根据新数据类型 newcursor_type 定义游标变量
  v_rec  employees%ROWTYPE;--定义行类型
BEGIN
--打开游标 v_cursor，根据传入的不同参数值，打开不同的查询（游标）
  IF choice = 1 THEN
    OPEN v_cursor FOR
      SELECT * FROM employees;
  ELSIF choice = 2 THEN
    OPEN v_cursor FOR
      SELECT * FROM departments;
```

```
  ELSIF choice = 3 THEN
    OPEN v_cursor FOR
      SELECT * FROM jobs;
  END IF;

  LOOP
    FETCH v_cursor
      INTO v_rec; -- 从游标变量 v_cursor 中取数据，放到变量 v_rec 中
    EXIT WHEN v_cursor%NOTFOUND;
    dbms_output.put_line(v_rec.last_name);--打印变量的内容
    --...
  END LOOP;
    CLOSE v_cursor;--关闭游标 v_cursor
END p_cursor;
```

OPEN-FOR 语句把一个游标变量和一个查询进行关联，并执行查询。除了游标的打开方式不相同以外，游标变量的使用（从游标中取数据、关闭游标等）与显示游标的使用区别不大。



19.3 一触即发——触发器介绍

触发器是使用 PL/SQL、Java 或者 C 语言编写的程序单元，存储在数据库服务器中，当一个表或一个视图被改变，或者数据库发生某些事件时，Oracle 自动点燃触发器，并执行触发器中的代码。只有在触发器中定义的事件发生时，触发器才被触发。触发器是自动执行的代码块，类似于存储过程，但是用户不能直接进行调用。

能够点燃触发器的事件有：DML 操作（INSERT、UPDATE、DELETE）、DDL 操作、系统事件（System Event）（如数据库的关闭/启动）、用户事件（User Event）（如用户的登录等）。

触发（点燃）触发器的语句叫**触发语句（Triggering Statement）**。如我们在 SQL*Plus 中执行一条 SQL 语句，这条 SQL 语句触发了一个触发器，我们就把这条 SQL 语句叫**触发语句**。



19.3.1 触发器的功能

触发器可以用于实现下列功能：

- 自动派生出新的列值
- 阻止无效的事务
- 强化 Oracle 的安全
- 启用复杂的商业逻辑
- 提供审计和日志记录
- 允许/限制对表的修改
- 强制数据一致性

- 收集表的统计信息

19.3.2 触发器的组成

如图 19-2 所示，一个触发器由三部分组成：触发事件或语句（Triggering Event or Statement）、触发限制（Trigger Restriction）和触发动作（Trigger Action）。

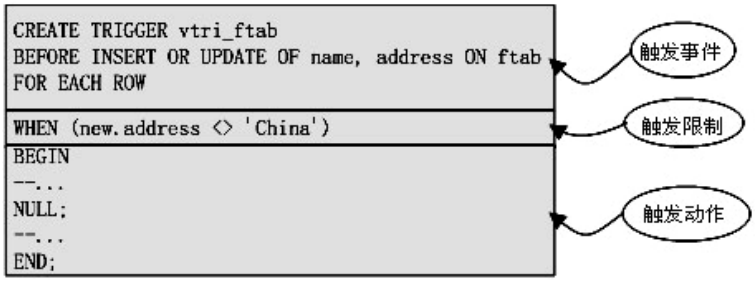


图 19-2 触发器的组成

1．触发事件或语句

触发事件或语句可以是 SQL 语句、数据库事件、用户事件。这些事件引起触发器被触发。

2．触发限制

触发限制为一布尔表达式，当触发器激发时该条件必须为 TRUE。如果布尔表达式的值为 Unknown 或者 False，“触发动作”将不会被执行。触发限制是用 WHEN 子句来指定。

3．触发动作

触发动作（Trigger Action）也叫触发主体（Trigger Body），为一个 PL/SQL 块（过程），由 SQL 语句和 PL/SQL 语句等组成。当触发限制为真（TRUE）时，它才被执行。在触发动作部分，可对当前字段的新值和旧值进行处理。

19.3.3 触发器的种类

触发器是基于（Table）、视图（View）、模式（Schema）、数据库（Database）的，据此，我们可以把触发器分成下面的几类。

- 行级触发器和语句级触发器（Row Trigger and Statement Trigger）
- BEFORE 和 AFTER 触发器（BEFORE and AFTER Trigger）
- 复合触发器（Compound Trigger）
- INSTEAD OF 触发器（INSTEAD OF Trigger）
- 系统级触发器（Triggers on System Event）
- 用户级触发器（Triggers on User Events）

当然，触发器还有其他分类方法，如 DDL 触发器、DML 触发器等。知道触发器的大致原理和使用足矣，不要对触发器的分类方法追根究底！

1．行级触发器和语句级触发器

行级触发器（Row Trigger），即触发机制是基于行的，当表中数据改变时，将触发行级触发器，改变一行数据，触发一次；改变 *n* 行数据，触发 *n* 次。

语句级触发器（Statement Trigger）是基于语句级的，当一条 SQL 语句改变数据时，无论这条 SQL 语句影响多少条记录，语句级触发器都只触发一次。SQL 语句每执行一次，语句级触发器就被触发一次。

2．BEFORE 和 AFTER 触发器

BEFORE 表示在触发语句运行前先运行“触发动作（Trigger Action）”。AFTER 表示在触发语句运行之后才运行“触发动作（Trigger Action）”。BEFORE 和 AFTER 适用于行级触发器和语句级触发器。

3．复合触发器

复合触发器是表上的触发器，它有 4 个时间点，可以让我们针对不同的时间点指定不同的处理动作。这 4 个时间点是：

- 在触发语句执行前（BEFORE STATEMENT）
- 在触发语句执行后（AFTER STATEMENT）
- 在每行记录被修改之前（BEFORE EACH ROW）
- 在每行记录被修改之后（AFTER EACH ROW）

触发语句必须是 DML。如果触发语句没有影响任何一行数据，并且也没有指定 BEFORE STATEMENT 和 AFTER STATEMENT 两个时间点，则触发器将不会被触发。

4．INSTEAD OF 触发器

有的视图，我们不能直接对其进行更新操作，但是我们可以在这种视图上建立触发器，利用触发器对视图的基表进行更新操作，这种类型的触发器就叫“INSTEAD OF 触发器”。利用 INSTEAD OF 触发器更新基表（students3）示例如下所示：

STEP 01 在基表 students3 上创建视图 v_stu。

```
SQL> CREATE OR REPLACE VIEW v_stu AS
  2   SELECT sum(id) allID,name
  3     FROM students3
  4     GROUP BY name;
```

STEP 02 在视图 v_stu 上创建 INSTEAD OF 触发器 tri_view。

```
SQL> CREATE OR REPLACE TRIGGER tri_view
  2   INSTEAD OF DELETE ON v_stu
  3   FOR EACH ROW
  4 BEGIN
```

```
  5    DELETE FROM students3 –从基表 students3 中删除数据
  6      WHERE id=11;
  7  END tri_view;
  8  /
```

STEP 03 查询基表 students3 中的数据。

```
SQL> SELECT * FROM students3;
```

SQL 语句的执行结果是：

```
       ID   NAME                CITY
---------   ------------------- --------------------
       12   KATE                HANGZHOU
       11   TOM                 HEBEI
```

从这里可以看到，点燃触发器 tri_view 以前，表中共有两条数据。

STEP 04 对视图 v_stu 进行更改，将点燃触发器 tri_view。

```
SQL> DELETE FROM v_stu;
```

STEP 05 触发器执行完成以后，查看基表中的数据。

```
SQL> SELECT * FROM students3;
       ID   NAME                CITY
----------  ------------------- --------------------
       12   KATE                HANGZHOU
```

从输出结果可以看出，基表中只有一条数据。ID=11 的这条记录已经被触发器自动删除。

5．系统事件触发器

系统事件触发器是基于数据库系统的触发器，系统事件触发器与表、视图没有关系，系统事件包括数据库启动（STARTUP）、关闭（SHUTDOWN）、服务器错误（SERVERERROR）、数据库角色改变（DB_ROLE_CHANGE）等。当这些事件发生时，就会触发系统事件触发器。可以通过系统事件触发器实现对数据库的审计。

6．用户事件触发器

用户事件包括用户登录（LOGON）数据库、用户退出数据库（LOGOFF）、用户执行 DDL 语句/DML 语句等。当这些事件发生时，会触发用户事件触发器。

19.3.4 触发动作（Trigger Action）的编码

本节将详细讲解触发动作。

1．触发动作的分类

触发动作，也叫触发主体（Trigger Body），它是一个 PL/SQL 块（也就是一段 PL/SQL 代码）或者是一个对子程序（存储过程和函数）的调用（使用 call 进行调用）。触发器主体是一个子程序调用的例子如下所示：

```
CREATE TRIGGER tri_p
BEFORE INSERT OR UPDATE OF id ON student
FOR EACH ROW
WHEN (new.id <> '00000')
CALL check_id(:new.id)
```

其中，触发器 tri_p 的主体是一个子程序调用，调用存储过程 check_id。



注：子程序包括 PL/SQL 子程序和 Java 子程序。

2．编写 PL/SQL 块

如果触发主体不是一个子程序调用，而是一个 PL/SQL 块，则需要对 PL/SQL 块进行编码。编写 PL/SQL 块涉及下面的内容。

在 PL/SQL 主体中访问列的值

在触发器中，列的值分为触发语句执行（尤其指修改）以前的值（新值, NEW）和执行以后的值（旧值, NEW）。如果触发器涉及嵌套表，还可以使用 PARENT 对父表进行引用。PL/SQL 新值和旧值的引用形式如下：

- :NEW.列名（引用新值）
- :OLD.列名（引用旧值）

需要注意的是，INSERT 没有旧值，只有新值。DELETE 没有新值，只有旧值。:NEW 与:OLD 只用于行级触发器。



示例　新值（NEW）旧值（OLD）应用——打印列 name 新值和旧值的内容。



创建测试表 tb_tri。

```
TABLE tb_tri (name VARCHAR2(20));
```



在测试表 tb_tri 上创建触发器 tb_tri_test。

```
CREATE OR REPLACE TRIGGER tb_tri_test
  AFTER UPDATE ON tb_tri
  FOR EACH ROW
```

```
BEGIN
  dbms_output.put_line('old value:' || :OLD.NAME);--打印列 name 修改以前的值
  dbms_output.put_line('new value:' || :NEW.NAME);--打印列修改以后的值
END;
```

STEP 03 往表 tb_tri 中插入测试数据。

```
INSERT INTO tb_tri VALUES ('万芳');
COMMIT;
```

STEP 04 设置 SQL*Plus 环境变量。

```
SQL> SET SERVEROUTPUT ON
```

STEP 05 触发（点燃）触发器。

```
SQL> UPDATE tb_tri SET name='刘芳'
  2  WHERE name='万芳';
old value:万芳
new value:刘芳
1 row updated
```

从输出结果可以看出，触发器被触发，并输出了列 name 的旧值和新值。

NEW 和 OLD 的重命名

为了防止命名冲突，我们可以对 NEW 和 OLD 进行重命名，使用选项 REFERENCING 进行重命名。这是 Oracle 提供的一个功能，不过很少用到。下面是 NEW 和 OLD 重命名的一个例子：

```
CREATE OR REPLACE TRIGGER tri_rename
BEFORE UPDATE ON tb_tri
REFERENCING NEW AS new_one  --对 NEW 进行重命名
            OLD AS old_one  --对 OLD 进行重命名
FOR EACH ROW
DECLARE
V_NEW VARCHAR2(20);
V_OLD VARCHAR2(20);
BEGIN
V_NEW:=:new_one.name;       --使用新名字对列进行引用
V_OLD:=:old_one.name;       --使用新名字对列进行引用
--...;
END;
```

判断触发的 SQL 语句类型（INSERTING、DELETING、UPDATING）

定义触发器时，在一个触发器中，可以指定多个触发语句，如：

```
CREATE TRIGGER tr1
BEFORE INSERT OR UPDATE OF…
…
```

本例中，INSERT 或者 UPDATE 都会点燃触发器，但有时候，我们需要知道到底是哪个语句点燃了触发器。Oracle 提供了三个判断条件（INSERTING、DELETING、UPDATING），用于判断点燃触发器的是 INSERT、UPDATE 还是 DELETE 操作。

```
CREATE OR REPLACE TRIGGER tri_DML
  AFTER INSERT OR UPDATE OR DELETE ON tb_tri
BEGIN
  IF UPDATING THEN
    --处理 1;
  ELSIF DELETING THEN
    --处理 2;
  ELSIF INSERTING THEN
    --处理 3;
  END IF;
END;
```

本例中，如果点燃触发器的是“更新操作（UPDATING）”，则执行“处理 1”；如果点燃触发器的是“删除操作（DELETING）”，则执行“处理 2”；如果点燃触发器的是“插入操作（INSERTING）”，则执行“处理 3”。



19.3.5 触发器的维护

本节将要讲述触发器的维护，强调的是实际操作。

1．创建触发器

使用 CREATE TRIGGER 命令创建触发器。用户可以在自己的模式下创建触发器，但是必须具有 CREATE TRIGGER 的权限。用户也可以在其他模式下创建触发器，但是必须具有系统权限 CREATE ANY TRIGGER。用户也可以创建系统级的触发器，但是必须具有 ADMINISTER DATABASE TRIGGER 权限。创建触发器的常规语法如下所示：

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER|INSTEAD OF} event
ON {table_name|view_name|DATABASE}
[FOR EACH ROW[WHEN condition]]
trigger_body
```


其中，trigger_name 指定触发器的名字；event 指定触发事件；condition 指定触发限制；trigger_body 定义触发动作（触发器主体）；table_name|view_name|DATABASE 基于表、视图或者数据库定义触发器。

创建行级触发器

```
CREATE OR REPLACE TRIGGER tri2
BEFORE INSERT ON TT.CT
FOR EACH ROW
```

```
BEGIN
NULL;
END;
```

其中，FOR EACH ROW 表示创建的触发器是行级触发器。

创建语句级触发器

```
CREATE OR REPLACE TRIGGER tri1
BEFORE INSERT ON TT.CT
BEGIN
NULL;
END;
```

没有使用选项 FOR EACH ROW，表示创建的是行级触发器。

创建 BEFORE 触发器

```
CREATE OR REPLACE TRIGGER tri_befor
BEFORE
DELETE OR INSERT OR UPDATE
ON tb_tri
BEGIN
NULL;
----
END;
```

其中，选项 BEFORE 表示创建的是 BEFOR 触发器。

创建 AFTER 触发器

创建 AFTER 触发器的 SQL 语句和创建 BEFORE 触发器的 SQL 语句类似，只是把 BEFORE 替换成 AFTER。

创建复合触发器

创建复合触发器的示例如下：

```
CREATE OR REPLACE TRIGGER tr_stuinfor
  FOR UPDATE OF name ON tb_tri COMPOUND TRIGGER
  a INT;--在触发器中定义变量

  BEFORE STATEMENT IS       --时间点 1，触发语句执行前
  BEGIN
    --...处理 1
  END BEFORE STATEMENT;

  BEFORE EACH ROW IS            --时间点 2，每条记录修改前
  BEGIN
    NULL;
    --... 处理 2
  END BEFORE EACH ROW;
```

```
  AFTER EACH ROW IS              --时间点 3，每条记录修改后
  BEGIN
     NULL;
     --... 处理 3
  END AFTER EACH ROW;

  AFTER STATEMENT IS             --时间点 4，触发语句执行后
  BEGIN
    NULL;
    --... 处理 4
  END AFTER STATEMENT;

  END tr_stuinfor;--触发器的结束点
```

本例中定义了 4 个时间点，针对不同的时间点，有不同的处理措施（处理 1、处理 2、处理 3、处理 4）。COMPOUND TRIGGER 表示创建的是复合触发器。

创建 INSTEAD OF 触发器

INSTEAD OF 触发器的创建，**请参考本书 19.3.3 节的内容。**

创建用户事件触发器

创建用户事件触发器 tri4。

```
CREATE OR REPLACE TRIGGER tri4
BEFORE CREATE ON tt.SCHEMA
BEGIN
RAISE_APPLICATION_ERROR (
num => -20001,
msg => 'Cannot CREATE object');
END;
```

当用户 tt 创建对象时点燃该触发器。SCHEMA 指示创建的是用户事件触发器。

创建系统事件触发器

创建系统事件触发器 log_errors，当数据库发生错误（SERVERERROR）后，点燃该触发器。


```
CREATE OR REPLACE TRIGGER log_errors AFTER SERVERERROR ON DATABASE
BEGIN
    --对错误进行处理
    NULL;
END;
```

其中，DATABASE 表示定义的是系统事件触发器。

系统事件也叫数据库事件，系统事件有 AFTER STARTUP、BEFORE SHUTDOWN、AFTER DB_ROLE_CHANGE、AFTER LOGON、BEFORE LOGOFF、AFTER SERVERERROR、AFTER SUSPEND。

2．禁用和启用触发器

所谓禁用触发器（Disabling Triggers），即使触发器失效，在任何条件下都不能触发；启用触发器（Enabling Triggers）即使触发器可用，可以被触发。

禁用触发器 LOGIN1。

```
SQL> ALTER TRIGGER LOGIN1 disable;
```

启动触发器 LOGIN1。

```
SQL> ALTER TRIGGER LOGIN1 enable;
```

3．删除触发器

使用 DROP TRIGGER 命令删除触发器。具有 DROP ANY TRIGGER 权限的用户可以删除任何触发器。如果要删除数据库级的触发器（ON DATABASE），用户还必须具有系统权限 ADMINISTER DATABASE TRIGGER。

删除触发器 login1。

```
SQL> DROP TRIGGER login1;
```



19.3.6 在触发器中，如何利用学生表 1 的值去更新学生表 2 的值

在本例中，如果学生表 students_A 中的学生姓名被修改，学生表 students_B 中的学生姓名也会被触发器自动修改，详细过程说明如下。



查看学生表 students_A 的结构。

```
SQL> DESC students_A
```

SQL 语句的执行结果是：

```
Name                NULL?   Type
------------------    --------    ----------------------------
ID                          NUMBER(38)
NAME                        VARCHAR2(20)
CITY                        VARCHAR2(20)
```



查看学生表 students_B 的结构。

```
SQL> DESC students_B
```

SQL 语句的执行结果是：

```
Name                NULL?   Type
------------------    --------    ----------------------------
ID                          NUMBER(38)
NAME                        VARCHAR2(20)
CITY                        VARCHAR2(20)
```

```
HEIGHT                         NUMBER(38)
```

STEP 03 在学生表 students_A 上创建触发器 tri_stufef。

```
SQL> CREATE OR REPLACE TRIGGER tri_stufef
  2  AFTER UPDATE ON students_A
  3  FOR EACH ROW
  4  DECLARE
  5  v_dept_id int;
  6  BEGIN
  7  UPDATE students_B set name=:new.name WHERE id=:old.id;
  8  END;
  9  /
Trigger created.
```

这个触发器将会根据学生的学号（ID）自动更新 students_B 中的学生姓名（Name）。

STEP 04 点燃触发器以前，表 students_A 中的数据如下所示。

```
SQL> SELECT * FROM students_A;
```

SQL 语句的执行结果是：

```
     ID     NAME    CITY
----------  --------    --------------------
       12   KATE        HANGZHOU
```

STEP 05 点燃触发器以前，表 students_B 中的数据如下所示。

```
SQL> SELECT * FROM students_B;
       ID   NAME    CITY            HEIGHT
------------ ------- --------------- --------------
       12   KATE        HANGZHOU
```

STEP 06 点燃触发器 tri_stufef。

```
SQL> UPDATE students_A set name='TOM' WHERE id=12;
1 row updated.
SQL> COMMIT;
COMMIT complete.
```


STEP 07 触发器 tri_stufef 点燃以后，表 students_A 发生变化。

```
SQL> SELECT * FROM students_A;
```

SQL 语句的执行结果是：

```
   ID       NAME    CITY
------------ ------ --------------------
       12   TOM     HANGZHOU
```

STEP 08 触发器 tri_stufef 点燃以后，表 students_B 发生变化。

```
SQL> SELECT * FROM students_B;
```

SQL 语句的执行结果是：

```
 ID         NAME    CITY        HEIGHT
----------- ------- ----------- -------- ----------
       12   TOM     HANGZHOU
```

从上面可以看出，触发器自动把表 students_B 中学号是 12 且姓名原来叫 KATE 的学生的名字改成了 TOM。



19.3.7 阻止触发语句的执行

通过过程 raise_application_error 报告异常，可以阻止触发语句的执行，就好像触发语句没有被数据库执行过一样。接下来，我们将通过一个例子演示如何组织触发语句的执行。

在本例中，如果插入的 ID 值小于 50，可以顺利插入；如果插入的 ID 值大于 50，触发语句将被阻止执行，详细过程说明如下。



在表 students3 上创建触发器 tri_auto。

```
SQL> CREATE OR REPLACE TRIGGER tri_auto
  2   before INSERT on students3
  3   FOR EACH ROW
  4  DECLARE
  5   sql_string VARCHAR2(100);
  6  BEGIN
  7   if :new.id>50 then --如果 ID 值大于 50，将会出现异常，阻止插入语句的执行
  8   raise_application_error(-20001,'can not INSERT user with id>50');
  9   END if;
 10  END tri_auto;
 11  /

Trigger created.
```

其中，raise_application_error()函数用于触发用户自定义的异常；–20001 是自定义的错误编号；'can not INSERT user with id>50'是自定义的错误信息。



点燃触发器以前，表中没有一条数据。

```
SQL> SELECT * FROM students3;
no rows SELECTed
```



点燃触发器 tri_auto。

```
SQL> INSERT INTO students3
  2  VALUES(51,'WW','shanghai');
INSERT INTO students3
          *
ERROR at line 1:
ORA-20001: can not INSERT user with id>50
ORA-06512: at "TEST.TRI_AUTO", line 5
```

```
ORA-04088: error during execution of trigger 'TEST.TRI_AUTO'
```

在这里，插入值 ID=51，ID 值大于 50，触发自定义异常。其中，ORA-20001: can not INSERT user with id>50 是我们上面自定义的异常。

STEP 04 点燃触发器以后，表中还是没有一条数据。

```
SQL> SELECT * FROM students3;
no rows SELECTed
```

从这里就可以看出，插入语句被阻止执行（因为 ID 是 51，大于 50）。



19.4 DDL 语句的救星——动态 SQL（Dynamic SQL）

在讲解动态 SQL 之前，我们先讲解一下静态 SQL 的局限。

■ 利用静态 SQL 在存储过程中创建表 tp——创建失败

STEP 01 创建存储过程 CREATE_table，在存储过程中创建表 tp。

```
SQL> CREATE OR REPLACE PROCEDURE CREATE_table
  2  as
  3  Pstring VARCHAR2(2000);
  4  BEGIN
  5  NULL;
  6  CREATE TABLE tp (id int,name VARCHAR2(20));
  7  END;
  8  /
Warning: Procedure created with compilation errors
```

STEP 02 使用 SHOW ERROR 命令显示存储过程创建时产生的错误信息。

```
SQL> SHOW ERROR;
Errors for PROCEDURE ITME.CREATE_TABLE:
LINE/COL ERROR
6/1  PLS-00103: 出现符号 "CREATE"在需要下列之一时：  ( begin case declare end
exception exit for goto if loop mod null pragma raise return    SELECT update
while with <an identifier>    <a double-quoted delimited-identifier> <a bind
variable> <<    continue close current delete fetch lock INSERT open rollback
savepoint set sql execute commit forall merge pipe purge
```

从语句的执行结果可以看出，存储过程创建出现错误，但是不知道错误的真正原因。从创建存储过程的脚本中，我们知道，该脚本中存在一条 DDL 语句（CREATE TABLE…），正是这条语句导致存储过程创建失败。

从这里暴露出静态 SQL 的局限性！

■ 利用动态 SQL 在存储过程中创建表 tp——创建成功

STEP 01 利用动态 SQL 创建存储过程。

```
CREATE OR REPLACE PROCEDURE CREATE_table
AS
Pstring VARCHAR2(2000);--定一个变量，用于存放创建表的 SQL 语句
BEGIN
Pstring:='CREATE TABLE tp (id int,name VARCHAR2(20))' ;--创建表的 SQL 语句
EXECUTE IMMEDIATE P_string; --使用 EXECUTE IMMEDIATE 语句执行动态 SQL 创建表
END;
```

 执行存储过程，创建表。

```
SQL> EXECUTE CREATE_table;
```

如果报告权限不足，需要使用命令 GRANT CREATE ANY TABLE To '用户名'把建表的权限授予执行存储过程的用户。

 用 DESC 命令检验表的存在性。

综合上面两个例子不难看出，动态 SQL 弥补了静态 SQL 的缺陷，静态 SQL 不能执行 DDL 语句，而动态 SQL 可以，所以，我们才说“DDL 语句的救星——动态 SQL（Dynamic SQL）”。



19.4.1 静态 SQL（Static SQL）

静态 SQL 语句（Static SQL Statements）是 PL/SQL 语句的一部分，这些语句可以在 PL/SQL 代码中被直接使用。在编译时，静态 SQL 语句已经被解析和验证过。也就是说，编译时，已经知道静态 SQL 语句的内容（包括执行的对象、数据类型、输入变量、输出变量等），这些 SQL 语句是“确定”的 SQL 语句。静态 SQL 的组成如表 19-8 所示。

表 19-8 静态 SQL 的组成

数据操作语言（DML）	事务控制语言（TCL）	SQL 函数（SQL Functions）	伪劣（Pseudocolumns）	SQL 运算符（SQL Operators）

- 数据操作语言（DML）包括 DELETE、INSERT、UPDATE。
- 事务控制语言（TCL）包括 COMMIT、ROLLBACK、SAVEPOINT、SETTRANSACTION。
- 伪劣（Pseudocolumns）包括 CURRVAL、NEXTVAL、LEVEL、ROWID、ROWNUM。
- SQL 运算符（SQL Operators）包括比较运算符（ALL、ANY、SOME、BETWEEN、EXISTS 、IN、IS NULL、LIKE）、集合运算符（INTERSECT、MINUS、UNION、UNION ALL）、行操作符（ALL、DISTINCT、PRIOR）。



注：静态 SQL 不支持 DDL 语句。



19.4.2 动态 SQL（Dynamic SQL）

动态 SQL 语句在编译时，并不知道 SQL 语句的内容，SQL 语句的内容“不确定”，只有在运行时，才建立、解析并执行 SQL 语句。利用动态 SQL，在存储过程中，可以动态创建表、视图、触发器等。

下面的情况适合使用动态 SQL：

- 编译时，SQL 语句的内容未知
- 静态 SQL 不支持的 SQL 语句（如 DDL 语句）



19.4.3 动态 SQL 和静态 SQL 的比较

一般情况下，静态 SQL 的执行效率比动态 SQL 高，因为静态 SQL 在编译时，已经提前做了某些工作：

- 编译提前验证了 SQL 语句涉及的数据对象和权限
- 编译提前建立了 SQL 语句涉及的对象之间的依赖性



19.4.4 动态 SQL 的书写

有两种方法（或者叫工具）可以用来书写动态 SQL：

- 本地动态 SQL（Native Dynamic SQL），用于建立和执行 SQL 语句。本地动态 SQL 使用 EXECUTE IMMEDIATE 命令来执行动态 SQL 语句。
- 包 DBMS_SQL 也可用于执行动态 SQL。

通常，本地动态 SQL 易于使用，性能较 DBMS_SQL 高。我们推荐使用本地动态 SQL。但是，有些功能本地动态 SQL 不能实现，只能用 DBMS_SQL 来完成。本书重点介绍本地动态 SQL。

1．EXECUTE IMMEDIATE 的使用

在讲述 EXECUTE IMMEDIATE 语句之前，我们必须明白什么是占位符（Placeholder）。占位符是 SQL 语句中的一个位置，绑定变量将为该位置提供值。占位符是未声明的标识符，它的名称并不重要，需要以冒号开头。每个占位符必须有一个相对应的绑定变量（位置对应）。

EXECUTE IMMEDIATE 语句的基本语法如下所示：

```
EXECUTE IMMEDIATE SQL_string
  [INTO {define_variable[, define_variable]... | record}]
  [USING [IN | OUT | IN OUT] bind_argument
     [, [IN | OUT | IN OUT] bind_argument]...];
```

其中，SQL_string 是 SQL 语句或者 PL/SQL 代码；INTO 指定存放返回值的普通变量

（define_variable）或者记录类型（record）的变量；bind_argument 指定绑定参数，IN 表示参数只输入，OUT 表示参数只输出。

本节的后续小节将会用到手机信息表 phone_infor，该表的结构如下：

```
SQL> DESC phone_infor
```

SQL 语句的执行结果是：

```
Name            Type         Nullable Default Comments
----------  ------------        -------- ------- --------
PHONE_NAME  VARCHAR2(20)        Y
PRODUCER    VARCHAR2(20)        Y
PRICE           NUMBER              Y
```

其中，PHONE_NAME 表示手机的名字；PRODUCER 表示手机生产商；PRICE 表示手机的价格。

🕮 没有占位符的 EXECUTE IMMEDIATE 语句

没有占位符的 EXECUTE IMMEDIATE 语句，指 EXECUTE IMMEDIATE 命令没有 USING 子句，简单地说，不会有值传入动态 SQL 语句中。如果 DML 语句含有占位符，那么在 EXECUTE IMMEDIATE 语句之后要带有 USING 子句（USING 用于指定绑定参数，也就是指定动态 SQL 语句的传入值）。

🕮 处理 DDL 语句

我们将通过一个存储过程（CreateTable）讲述如何使用 EXECUTE IMMEDIATE 执行 DDL 语句。根据传入参数的不同，存储过程 CreateTable 会创建不同的表。

STEP 01 创建存储过程 CreateTable。

```
CREATE OR REPLACE PROCEDURE CreateTable(tablename VARCHAR2)
IS
BEGIN
  EXECUTE IMMEDIATE  'CREATE TABLE ' || tablename || ' ( name VARCHAR(20))';
END;
```

存储过程 CreateTable 的输入参数是表的名字（tablename），根据表名的不同，该存储过程动态创建不同的表。EXECUTE IMMEDIATE 语句用于执行动态 SQL，tablename 是存储过程的传入参数，表示表的名字。

STEP 02 执行存储过程 CreateTable。

```
SQL> EXECUTE CreateTableProc('TXX');
```

该存储过程将创建表 TXX。

STEP 03 验证存储过程是否创建了表 TXX。

```
SQL> DESC TXX
Name Type        Nullable Default Comments
---- ------------ -------- ------- --------
NAME VARCHAR2(20) Y
```

从上面可以看出，表 TXX 已经创建。

有的朋友会认为，使用**绑定参数**来动态传递表的名字，也能动态创建表。错也！请看下面的例子：

```
CREATE OR REPLACE PROCEDURE CreateTableProc(tablename VARCHAR2)
IS
SQLstring VARCHAR2(1000);
BEGIN
   SQLstring:='CREATE TABLE :tb ( name VARCHAR(20))';--这里使用了占位符
   EXECUTE IMMEDIATE  SQLstring
   USING tablename;--这里使用了绑定参数
END;
```

执行命令 EXECUTE CreateTableProc('TXX')，将会报告错误“ORA-00903: invalid table name”，所以，我们必须注意，绑定参数不能是模式对象的元素（如表名、列名、数据类型等），绑定参数只能是值、变量或者表达式。用 DDL 语句动态创建对象时，推荐使用连接运算符（||），不推荐使用绑定参数。

处理 TCL 语句

我们可以在 PL/SQL 代码中直接执行 COMMIT 和 ROLLBACK 等事务控制语句，也可以在动态 SQL 中执行事务控制语句，两者实现相同的效果。

```
DECLARE
SQLstring VARCHAR2(1000);
BEGIN
  SQLstring:='COMMIT';
  --插入数据
  EXECUTE IMMEDIATE 'INSERT INTO phone_infor values(''E68'','' 诺 基 亚 '',444)';    EXECUTE IMMEDIATE SQLstring;--提交数据
END;
```

本例往表中插入一条数据，并进行提交。也可把 EXECUTE IMMEDIATE SQLstring 语句替换成 COMMIT 语句，它们实现相同的功能。

处理 DML 语句

本节将讨论如何利用动态 SQL 执行 DML 语句。我们将分成下面几种情况进行讨论。

■ 执行普通的 DML 语句——不带占位符（不带 USING 子句）

```
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO phone_infor values(''E68'','' 诺 基 亚 '',400)';
  COMMIT;
END;
```

本例中，没有值传入动态 SQL 中。

■ 执行特殊的 DML 语句——带占位符（使用 USING 子句给 DML 语句传入值）

```
DECLARE
  v_phone_name VARCHAR2(20):='C510c';
  v_producer   VARCHAR2(20):='爱立信';
  v_price     NUMBER:=1201;
```

```
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO phone_infor values(:a, :b, :c)'
    USING v_phone_name, v_producer, v_price;
  COMMIT;
END;
```

本例中，有值被传入动态 SQL 中。

其中，:a、:b、:c 是占位符，占位符以冒号开始，其名字无关紧要。USING 指定绑定参数列表，绑定参数与占位符在位置上一一对应，本例中，:a 对应 v_phone_name；:b 对应 v_producer；:c 对应 v_price。如果指定占位符，必须使用 USING 子句。

■ 执行有返回值的 DML

当执行 DML 后，可以返回受影响的行的值。特别注意的是，如果只有一行受到影响，返回值存放在 PL/SQL 变量中；如果有多行受到影响，则返回值放在数组中。从 DML 返回值的例子如下所示。本例将把手机价格是 0 的手机价格更新成 1201，并返回手机的名字（phone_name）、手机生产商（producer）及价钱（price）。

```
DECLARE
  SQLstring    VARCHAR2(1000);
  v_phone_name VARCHAR2(20);
  v_producer   VARCHAR2(20);
  v_price     NUMBER :=0;
BEGIN
   SQLstring := 'UPDATE phone_infor SET price = 1201 WHERE PRICE=:a RETURNING
    phone_name,producer,price INTO :b,:c,:d';
   EXECUTE IMMEDIATE SQLstring USING v_price RETURNING INTO
       v_phone_name,v_producer,v_price;
   COMMIT;
   DBMS_OUTPUT.PUT_LINE(v_phone_name||'  '||v_producer||'  '||v_price);
END;
```

执行上述代码将输出如下结果：

```
C510c  爱立信  1201
```

对于有返回值的 DML 语句，输入绑定变量（IN）放在 **USING** 之后，输出绑定变量（OUT）放在 **RETURNING INTO** 之后。

对于占位符，:a 对应 USING 子句之后的 v_price。:b 对应列 phone_name 和绑定参数 v_phone_name。:c 对应列 producer 和绑定参数 v_producer。: d 对应列 price 和绑定参数 v_price。

在动态 SQL 中调用存储过程或者函数

■ 在动态 SQL 中调用存储过程

我们将通过一个完整的例子演示在动态 SQL 中如何调用存储过程。

STEP 01 创建存储过程。

```
CREATE OR REPLACE PROCEDURE p_call(p1 IN VARCHAR2 := NULL) AS
BEGIN
  DBMS_OUTPUT.PUT_LINE(p1||'is from PROCEDURE p_call');--存储过程将打印一些
内容
```

```
END p_call;
```

STEP 02 在动态 SQL 中，调用存储过程。

```
BEGIN
  EXECUTE IMMEDIATE 'CALL p_call(''hihihihi'')';
END;
```

上面代码执行的结果是：

```
hihihihiis from PROCEDURE p_call
```

其中，CALL 是调用存储过程的命令；p_call 是存储过程的名字。hihihihi 是存储过程的传入参数。

■ 在动态 SQL 中调用函数

需要注意的是，既然是调用函数，就一定有返回值。在动态 SQL 中调用函数的例子如下。

STEP 01 创建函数 F_Call。

```
CREATE OR REPLACE FUNCTION F_Call(p1 IN VARCHAR2)
  RETURN VARCHAR2 AS
BEGIN
  RETURN p1||'is from FUNCTION F_Call';
END F_Call;
```

STEP 02 通过动态 SQL 调用函数 F_Call。

```
DECLARE
 V_return VARCHAR2(50);--定义变量，用于存放函数的返回值
BEGIN
 EXECUTE IMMEDIATE 'CALL F_Call(''hihihihi'') INTO:functiong_Result'
 USING OUT V_return;
 DBMS_OUTPUT.PUT_LINE(V_return);--打印函数的返回值
END;
```

上面代码执行的结果如下所示：

```
hihihihiis from FUNCTION F_Call
```

其中，CALL 表示函数调用；F_Call 是函数的名字；hihihihi 是函数的参数；functiong_Result 是占位符，与 V_return 相对应；V_return 用于存放函数的结果。需要注意的是，V_return 必须为输出参数（OUT）。

处理单行查询

利用动态 SQL，在 PL/SQL 中，可以对表进行查询（尤其是动态查询）。通过 INTO 子句，可以把查询结果保存在变量（或者记录）中。需要注意的是：这里最多只能返回一行记录。

示例：求诺基亚公司最贵的手机的价格。

```
DECLARE
  v_maxprice NUMBER;
  v_producer VARCHAR2(20) := '诺基亚';
BEGIN
  EXECUTE IMMEDIATE 'SELECT max(price) from phone_infor where producer=:a'
    INTO v_maxprice --存放查询的返回值
```

```
    USING v_producer;-- v_producer 是绑定参数，对应占位符:a
  dbms_output.put_line(v_maxprice);--把查询结果打印出来
END;
```

在单行查询中，我们需要注意输出绑定变量（OUT）和输入绑定变量（IN）的位置，输出绑定变量（OUT）放在 INTO 之后；输入绑定变量（IN）放在 USING 之后。

处理多行查询

如果动态 SQL 语句是一个查询语句，并且返回多行记录，可以使用下面两种方法进行处理。

- 使用带有子句 BULK COLLECT INTO 的 EXECUTE IMMEDIATE 语句
- 使用 OPEN-FOR、FETCH、CLOSE 语句
- 使用 BULK COLLECT INTO 处理多行查询

采用 BULK COLLECT 可以将查询结果一次性地加载到集合中，而不是通过游标一条条地处理，这样的处理速度很慢，可以在 SELECT INTO、FETCH INTO、RETURNING INTO 语句中使用 BULK COLLECT。



注：在使用 BULK COLLECT 时，所有的 INTO 变量都必须是集合(COLLECTION) 类型。

示例：BULK COLLECT 的应用。

STEP 01 查看表 phone_info 的结构。

```
SQL> DESC phone_infor
```

SQL 语句的执行结果是：

```
Name            Type         Nullable Default Comments
----------  ------------        -------- ------- --------
PHONE_NAME  VARCHAR2(20)        Y
PRODUCER    VARCHAR2(20)        Y
PRICE           NUMBER          Y
```

STEP 02 根据表 phone_infor 的结构创建对象类型 phone_type。

```
CREATE OR REPLACE TYPE phone_type AS OBJECT
(
  phone_name VARCHAR2(20),
  producer   VARCHAR2(20),
  price      NUMBER
);
```

表 phone_infor 的结构**请参考本书 19.4.4 节的内容**。

STEP 03 创建表类型 phone_tab。

```
CREATE OR REPLACE TYPE phone_tab AS TABLE OF phone_type;
```

无论是 phone_tab 还是 phone_type，都是新的数据类型。

STEP 04 把查询结果整体放到变量 v_rec 中。

```
DECLARE
  v_rec phone_tab;--定义表类型变量的 v_rec， phone_tab 是一种数据类型
BEGIN
  EXECUTE IMMEDIATE 'SELECT phone_type(phone_name,producer,price) from
```

```
phone_infor'        BULK  COLLECT  INTO v_rec;
    --输出表 v_rec 的内容
      FOR i IN v_rec.FIRST .. v_rec.LAST LOOP
        dbms_output.put_line(v_rec(i).phone_name || ' ' || v_rec(i)
                          .producer || ' ' || v_rec(i).price);
      END LOOP;
    END;
```

在 EXECUTE IMMEDIATE 语句中出现 phone_type，这是进行数据类型转换，把查询结果的类型转换成 phone_type。BULK COLLECT INTO 表示把查询结果批量存放到变量 v_rec 中。FIRST 表示表的第一个元素，LAST 表示表的最后一个元素。

使用 OPEN-FOR、FETCH、CLOSE 语句处理多行查询

联合使用 OPEN-FOR、FETCH、CLOSE 语句，可以完成多行查询的处理。OPEN-FOR 表示打开游标；FETCH 用于从游标中取数据；CLOSE 用于关闭游标。利用 OPEN-FOR、FETCH、CLOSE 语句处理多行查询的例子如下：

本例返回价格高于 500 的手机的信息。

```
    DECLARE
      TYPE ref_type IS REF CURSOR;--定义游标指针
      v_cursor  ref_type;--定义游标变量
      v_record  phone_infor%ROWTYPE;--定义记录类型的变量
      sqlstring VARCHAR2(200);-- 变量 sqlstring 用于存放 SQL 语句
    BEGIN
      sqlstring := 'SELECT * FROM phone_infor WHERE price > :p';
      OPEN v_cursor FOR sqlstring --打开游标（执行动态 SQL）
        USING 500;--向动态 SQL 中传入值 500，500 对应占位符:p
      LOOP
        FETCH v_cursor
          INTO v_record;--从游标中取一条记录，存放到变量 v_record 中
    --输出记录的内容
        dbms_output.put_line(v_record.phone_name || ' ' || v_record.producer ||
' ' ||v_record.price);
        EXIT WHEN v_cursor%NOTFOUND;
      END LOOP;
      CLOSE v_cursor;--关闭游标
    END;
```

其中，v_record.phone_name 表示组成记录 v_record 的列 phone_name。

2．DBMS_SQL 的使用

DBMS_SQL 包提供一个接口，用于执行动态 SQL（包括 DDL 和 DML）。DBMS_SQL 定义了一个实体叫游标 ID，游标 ID 是一个 PL/SQL 整型数，通过游标 ID，可以对游标进行操作。DBMS_SQL 包和本地动态 SQL 在功能上有许多重叠的地方，但是有的功能只能通过本地动态 SQL 实现，而有的功能只能通过 DBMS_SQL 实现。

包 DBMS_SQL 提供了一系列的过程和函数，用于协同执行动态 SQL，这些函数和过程是：

■ 函数

- OPEN_CURSOR，打开一个动态游标，并返回一个整型。
- EXECUTE（c IN INTEGER），执行游标，并返回处理的行数（对 INSERT、DELETE、UPDATE 有意义，对 SELECT、DDL 语句没有意义）。
- FETCH_ROWS（c IN INTEGER），循环从游标中取数据。

■ 过程

- CLOSE_CURSOR（c IN OUT INTEGER）。关闭游标，参数为 OPEN_CURSOR 所打开的游标。
- PARSE（c IN INTEGER, statement IN VARCHAR2, language_flag IN INTEGER），对动态游标所提供的 SQL 语句进行解析，参数 c 表示游标，statement 为 SQL 语句，language_flag 为解析 SQL 语句所用版本，一般有 V6、NATIVE 和 V7。如果被解析的是 DDL 语句，则解析完成后被立即执行。
- DEFINE_COLUMN（c IN INTEGER, position IN INTEGER, column ANY DATATYPE, [column_size IN INTEGER]），定义从游标中选出的列，其中 c 为游标，position 为对应动态 SQL 中的位置（从 1 开始），column 为该值所对应的变量，可以为任何类型，column_size 定义列的宽度，如果列的类型为字符型，必须定义宽度。这个语句只用于带有 SELECT 语句。
- COLUMN_VALUE（c IN INTEGER, position IN INTEGER, VALUE），将所取得的游标数据赋值到相应的变量，c 为游标，position 为位置，VALUE 则为对应的变量。
- BIND_VARIABLE（c IN INTEGER, name IN VARCHAR2, VALUE），定义动态 SQL 语句（DML）中所对应字段的值，c 为游标，name 为字段名称，VALUE 为字段的值。

🕮 利用 DBMS_SQL 执行 DDL 语句

DBMS_SQL 执行 DDL 语句的过程是：打开游标→SQL 语句解析→关闭游标。本节中存储过程 CreateTable 和存储过程 CreateTable 的功能完全相同，都是动态创建表。使用 DBMS_SQL 执行 DDL 语句的详细过程说明如下。

STEP 01 创建存储过程 CreateTable。

```
CREATE OR REPLACE PROCEDURE CreateTable(tablename VARCHAR2)
IS
SQL_string VARCHAR2(1000);--存放 SQL 语句
```

```
    V_cur integer;--定有整型变量，用于存放游标
    BEGIN
    SQL_string:='CREATE TABLE ' || tablename || ' ( name VARCHAR(20))';
    V_cur :=dbms_sql.open_cursor;--打开游标
    dbms_sql.parse(V_cur,SQL_string,DBMS_SQL.NATIVE);--解析并执行 SQL 语句
    dbms_sql.close_cursor(V_cur);--关闭游标
    END;
```

STEP 02 执行存储过程 CreateTable，创建表 MYTB。

```
SQL> EXECUTE CreateTable('MYTB')
```

STEP 03 验证表 MYTB 是否成功创建。

```
SQL> DESC MYTB
Name Type        Nullable Default Comments
---- ------------ -------- ------- --------
NAME VARCHAR2(20) Y
```

利用 DBMS_SQL 执行 SELECT 语句

利用 DBMS_SQL 执行 SELECT 语句会大致经历下面的过程：打开游标→解析动态 SQL→绑定输入参数→定义列→执行动态 SQL→从游标中取数据→把结果值放入变量中→关闭游标。

本节例子的功能和使用 OPEN-FOR、FETCH、CLOSE 语句处理多行查询的例子的功能一样，都是用于返回价格高于 500 的手机的信息。详细的代码如下所示：

```
    DECLARE
      v_cursor     NUMBER; --游标 ID
      sqlstring    VARCHAR2(200); --用于存放 SQL 语句
      v_phone_name VARCHAR2(20); --手机名字
      v_producer   VARCHAR2(20); --手机生产商
      v_price      NUMBER := 500; --手机价钱
      v_count      INT; --在这里无意义，只是存放函数返回值
    BEGIN
      --:p 是占位符
      --SELECT 语句中的第 1 列是 phone_name，第 2 列是 producer，第 3 列是 price
      sqlstring := 'SELECT phone_name,producer,price  FROM phone_infor WHERE
price > :p';
      v_cursor  := dbms_sql.open_cursor; --打开游标;
      dbms_sql.parse(v_cursor, sqlstring, dbms_sql.native); --解析动态 SQL 语句;

      --绑定输入参数，v_price 的值传给:p
      dbms_sql.bind_variable(v_cursor, ':p', v_price);

      --定义列，v_phone_name 对应 SELECT 语句中的第 1 列
      dbms_sql.define_column(v_cursor, 1, v_phone_name, 20);
      --定义列，v_producer 对应 SELECT 语句中的第 2 列
      dbms_sql.define_column(v_cursor, 2, v_producer, 20);
```

```
    --定义列，v_price 对应 SELECT 语句中的第 3 列
    dbms_sql.define_column(v_cursor, 3, v_price);

    v_count := dbms_sql.EXECUTE(v_cursor); --执行动态 SQL 语句。

    LOOP
      --从游标中把数据检索到缓冲区（BUFFER）中，缓冲区的值只能被函数 COLUMN_VALUE（）
所读取
      EXIT WHEN dbms_sql.fetch_rows(v_cursor) <= 0;

      --函数 column_value()把缓冲区的列的值读入相应变量中。
      --第 1 列的值被读入变量 v_phone_name 中
      dbms_sql.column_value(v_cursor, 1, v_phone_name);
      --第 2 列的值被读入变量 v_producer 中
      dbms_sql.column_value(v_cursor, 2, v_producer);
      --第 3 列的值被读入变量 v_price 中
      dbms_sql.column_value(v_cursor, 3, v_price); --打印变量的值
      dbms_output.put_line(v_phone_name || '  ' || v_producer || '  ' ||
                           v_price);
    END LOOP;

    dbms_sql.close_cursor(v_cursor); --关闭游标。
  END;
```

利用 DBMS_SQL 执行 DML 语句

通过 DBMS_SQL 执行 DML 语句的过程大致是：打开游标→解析动态 SQL→绑定输入参数→执行动态 SQL→关闭游标。通过 DBMS_SQL 执行 DML 语句的例子如下：

```
  DECLARE
    v_cursor    NUMBER; --游标 ID
    sqlstring   VARCHAR2(200);   --用于存放 SQL 语句
    v_phone_name VARCHAR2(20);   --手机名字
    v_producer   VARCHAR2(20);--手机生产商
    v_price     NUMBER;       --手机价钱
    v_count     INT;      --被 DML 语句影响的行数
  BEGIN
    sqlstring   := 'INSERT INTO phone_infor values(:a,:b,:c)';-- :a,:b,:c
是占位符
    v_phone_name := 'W995';
    v_producer   := '索尼爱立信';
    v_price     := 999;
    v_cursor    := dbms_sql.open_cursor; --打开游标;
    dbms_sql.parse(v_cursor, sqlstring, dbms_sql.native); --解析动态 SQL 语句
     --绑定输入参数
   dbms_sql.bind_variable(v_cursor, ':a', v_phone_name);--v_phone_name 的值
传递给:a
    dbms_sql.bind_variable(v_cursor, ':b', v_producer);  --v_producer 的值传
```

```
递给:b
      dbms_sql.bind_variable(v_cursor, ':c', v_price);--v_price 的值传递给:c
      v_count := dbms_sql.EXECUTE(v_cursor);--执行动态 SQL
      dbms_sql.close_cursor(v_cursor);--关闭游标
      dbms_output.put_line('INSERT '||to_char(v_count)||' row');--打印有多少行被插入
      COMMIT;
    END;
```



本章总结

本章主要介绍 PL/SQL 程序开发。PL/SQL 程序开发和其他语言的开发有很多相似之处（如变量声明、控制结构）。如果你想深刻理解 PL/SQL 程序开发，建议你多做实验，因为有些概念难于理解，一旦你把这些概念融入到实验中，就会真正理解这些概念。如果你想做数据库的一些复杂应用，建议你好好研究一下动态 SQL。

第 20 章　存放在数据库服务器中的代码

存放在数据库服务器中的代码有存储过程、函数、包。本章将对它们进行介绍。



20.1　存储 PL/SQL 单元

存储 PL/SQL 单元（Stored PL/SQL Units）就是子程序（存储过程、函数）和包。简单地说，存储过程、函数和包统称为存储 PL/SQL 单元。存储 PL/SQL 单元有下面的特征：

- 有一个名字。
- 可以接受参数，有返回值（函数）。
- 存储在数据字典中。
- 能够被许多用户调用。

子程序包括存储过程和函数，如果一个子程序属于一个包，则这个子程序被称为包子程序（Package Subprogram）；如果一个子程序不属于任何一个包，则这个子程序被称为独立子程序（Standalone Subprogram）。



20.2　子程序的参数

调用子程序（存储过程和函数）时，可以向子程序中传递参数，子程序的参数有三个属性：参数的名字、参数的模式、参数的数据类型。

1．参数的名字

参数的名字必须为合法的 PL/SQL 标识符。

2．参数的模式

参数模式定义了参数的行为，参数有三种模式：IN、OUT 和 IN OUT。IN 表示参数只用于传入数据，参数的值不能被修改，我们把这样参数叫做“传入参数”。OUT 主要用于把值返回给调用者，我们把这样参数叫做“传出参数”。IN OUT 表示把值传入子程序中，值被修改以后，返回给调用者。IN OUT 模式实现了 IN 和 OUT 两个功能。IN 和 OUT 区别的例子如下。

创建函数 f1，f1 有两个参数 x 和 y，x 的模式是 IN，y 的模式是 OUT。

```
SQL> CREATE OR REPLACE FUNCTION f1(x IN NUMBER,y OUT number) RETURN number
  2    IS
  3  BEGIN
  4    x:=1;
  5    y:=2;
  6    RETURN 3;
  7  END;
  8  /
警告: 创建的函数带有编译错误。
SQL> show error;
4/3     PLS-00363: 表达式 'X' 不能用作赋值目标
```

从这里可以看出，模式为 IN 的参数 x 的值不能被改变（只用作传入值使用），而模式为 OUT 的参数 y 的值却可以改变。

3．参数的数据类型

参数的数据类型可以直接获得也可以间接获得。所谓直接获得，就是直接指定参数的数据类型；所谓间接获得，就是参数的数据类型从表或者列继承而来（也许我们并不知道列的实际数据类型）。使用%TYPE 和 %ROWTYPE 间接获得参数的数据类型。

■ 使用%TYPE 指定参数的数据类型

```
CREATE OR REPLACE PROCEDURE p_deal(p1 IN phone_infor.producer%TYPE)
...
```

使用%TYPE，参数的数据类型从列中继承，也就是参数的数据类型和列的数据类型相同，如果列的数据类型发生改变，参数的数据类型也会自动发生改变，这种机制可以增加程序的可维护性。

本例中，参数 p1 的数据类型和表 phone_infor 中列 producer 的数据类型相同。

■ 使用%ROWTYPE 指定参数的数据类型

```
CREATE OR REPLACE PROCEDURE p_deal(p2 OUT phone_infor%ROWTYPE)
...
```


%ROWTYPE 会定义一个记录类型，这个记录类型包含表中所有的列。

本例中，参数 p1 为记录类型，这个记录类型含有表 phone_infor 中所有的列。

可以把 PL/SQL 表、记录作为一个参数传递到子程序中。

20.3 存储过程（Procedure）

存储过程属于子程序。存储过程是编译好且存储在数据库服务器中的程序代码。存储过程有自己的名字及参数。但是，存储过程没有返回值。一个存储过程就是一组命名的 PL/SQL 语句。存储过程有如下优点：

- 存储过程存放在数据库服务器中，且已经编译好，并在服务器端执行，因此执行效率高。
- 增强数据安全性。存储过程把商业逻辑固化在存储过程中，并隐藏数据。
- 增强程序开发的灵活性和模块化。
- 可减少网络通信量。



20.3.1 存储过程的创建

存储过程也由 3 部分组成：定义部分、执行部分、异常处理部分。

如果只想在自己的模式中创建存储过程，则只需具有 CREATE PROCEDURE 权限；如果想在其他模式下创建存储过程，用户必须具有 CREATE ANY PROCEDURE 权限。拥有 ALTER ANY PROCEDURE 权限的用户可以替换其他模式的存储过程。

创建存储过程的语法如下所示：

```
CREATE [ OR REPLACE] PROCEDURE [schema.]procedure_name [parameter_list]
{AS|IS}
declaration_section
BEGIN
executable_section
EXCEPTION
exception_section
END [procedure_name]
```

其中，OR REPLACE 表示如果存储过程已经存在，则替换已经存在的存储过程。schema 表示模式名，将在指定的模式下创建存储过程。procedure_name 表示存储过程的名字。parameter_list 表示存储过程的参数，参数有三种模式：IN 表示调用存储过程时，必须给参数指定一个输入值，参数的默认模式是 IN；OUT 表示在存储过程中将一个值传给这个参数，存储过程执行完成后，可以访问这个值；IN OUT 是前面两种的组合。declaration_section 表示存储过程的定义部分，主要用于定义变量。executable_section 表示执行部分，含有一系列的 PL/SQL 语句。exception_section 表示异常处理部分，用于异常处理。

示例：创建存储过程 pro1。

```
SQL> CREATE PROCEDURE pro1(id NUMBER)
  2  AS
  3  v1 NUMBER;
  4  BEGIN
  5  INSERT INTO ct
  6  VALUES(v1,'SSSS',2,4);
  7  COMMIT;
  8  END;
```

其中，id 是存储过程的参数，参数的数据类型是 NUMBER；v1 是变量，变量的数据类型是 NUMBER；BEGIN 到 END 之间是存储过程的执行部分。

20.3.2 存储过程的调用

在不同的环境下，存储过程的调用不一样。

■ SQL*Plus 中执行存储过程

在 SQL*Plus 中，通过 SQL*Plus 命令 EXECUTE 执行存储过程。要想执行存储过程，用户必须具有该存储过程上的 EXECUTE 权限。执行存储过程的语法如下所示：

```
EXECUTE procedure_name(argument_list)
```

其中，procedure_name 表示存储过程的名字；argument_list 为存储过程的参数值列表。

示例：在 SQL*Plus 中直接执行存储过程 pro1。

```
SQL> EXECUTE pro1(2);
```

其中，EXECUTE 是关键字，表示执行存储过程；pro1 是存储过程的名字；2 是存储过程的参数值。

■ 在块中、函数、存储过程中执行存储过程。

```
SQL> DECLARE
  2  BEGIN
  3  pro1(2);
  4  END;
```

在块、函数、存储过程中执行存储过程的方式相同，都是把存储过程作为一个单独的语句执行。

20.3.3 存储过程的编译

如果存储过程包含的对象发生改变，可能使存储过程变得无效，这时，需要对存储过程进行重新编译，以使存储过程变得有效。使用 ALTER PROCEDURE ... COMPILE 命令编译存储过程。要想编译其他用户的存储过程，执行编译的用户必须被授予 ALTER ANY PROCEDURE 权限。

示例：编译存储过程 pro1。

```
SQL> ALTER PROCEDURE pro1 COMPILE;
```

其中，pro1 是存储过程的名字。

20.3.4 存储过程的删除

使用 DROP PROCEDURE 命令删除存储过程。但是，如果一个存储过程是包的一部分，则不能使用该命令删除包中的存储过程。要想删除其他用户创建的存储过程，执行删除的用户必须被授予 DROP ANY PROCEDURE 权限。

示例：删除存储过程 pro1。

```
SQL> DROP PROCEDURE pro1;
```

不能使用这种方法删除包中的存储过程。



20.3.5 存储过程使用技巧

本节主要展示存储过程应用中的一些技巧。

1．在存储过程中创建表

利用动态 SQL，可以在存储过程中动态创建表。在存储过程中创建表的原理和例子，**请参考本书 19.4.4 节的内容**。

2．在存储过程中创建触发器

利用动态 SQL（EXECUTE IMMEDIATE 语句或者 DBMS_SQL 包），可以在存储过程中创建动态触发器。接下来，我们将通过一个完整的例子告诉大家，如何在存储过程（CREATE_trigger）中创建触发器（本例假设已经存在表 students2）。


创建存储过程 CREATE_trigger。

```
CREATE OR REPLACE PROCEDURE CREATE_trigger
AS
P_string VARCHAR2(2000);--这个变量用于存放创建触发器 tri_row 的 SQL 语句
BEGIN
P_string:='
CREATE OR REPLACE TRIGGER tri_row
AFTER DELETE ON students2
DECLARE
v3 INTEGER;
BEGIN
SELECT COUNT(*) INTO v3 FROM students2;
END;';--把创建触发器 tri_row 的 SQL 语句存在变量 P_string 中
EXECUTE IMMEDIATE p_string;--这里真正执行创建触发器的 SQL 语句
END;
```


执行存储过程，创建触发器 tri_row。

```
SQL> EXECUTE CREATE_trigger;
```


验证触发器 tri_row 创建是否成功。

```
SQL>  SELECT OWNER,TRIGGER_NAME
  2   FROM dba_triggers
  3   WHERE owner='TEST' and trigger_name='TRI_ROW'
OWNER                          TRIGGER_NAME
------------------------------ --------------------------
```

```
TEST                                TRI_ROW
```

其中，TEST 是触发器的创建者。

从输出结果可以看出，触发器 tri_row 创建成功。

3．无法删除存储过程

当存储过程正在执行时，如果试图删除、修改、编译存储过程，等待一段时间后，Oracle 将会报告错误“ORA-04021 timeout occurred while waiting to lock object stringstringstringstringstring”。如果出现这样的错误，请按照下面的步骤解决该错误。

STEP 01 等待一段时间，重新尝试删除、修改或者编译存储过程。

如果再次删除、修改或者编译视图失败，需要继续执行下面的步骤。

STEP 02 查看哪些会话正在执行该存储过程。

```
SELECT a.sid,a.serial#,b.sql_text
  FROM v$session a, v$sql b
 WHERE a.sql_hash_value = b.hash_value
   AND a.sql_address = b.address
   AND b.sql_text LIKE '%p_lock%'
```

其中，p_lock 是存储过程的名字。这条语句的执行结果如下所示：

```
SID SERIAL# SQL_TEXT
---- ---------- --------------------------------------------------
128 340 SELECT a.sid,a.serial#,b.sql_text FROM v$session a, v$sql b WHERE
a.sql_hash
132 274 BEGIN p_lock; END;
```

这里需要自己小心区别，到底是哪个会话在执行该存储过程，不要错杀其他进程。从查询结果可以看出，SID 是 **132**、SERIAL#是 **274** 的会话正在执行存储过程 p_lock。

STEP 03 杀死正在执行存储过程 p_lock 的会话。

```
ALTER SYSTEM KILL SESSION '132,274';
```

如果 ALTER SYSTEM KILL SESSION 命令执行失败，需要继续执行后续步骤，否则到此为止。

STEP 04 根据 SID 得到 SPID。

```
SELECT spid
  FROM v$process p, v$session s
 WHERE s.paddr = p.ADDR and sid = 132
```

这条语句的输出结果是：

```
 SPID
------------------------
1572
```

STEP 05 使用操作系统命令杀掉进程。

```
kill -9 1572（kill 是 UNIX 命令）
```

STEP 06 再次尝试删除、修改或者编译存储过程。

4．执行存储过程没有输出

在 SQL*Plus 中执行存储过程，如果没有 SERVEROUTPUT 打开，dbms_output.put_line 不会向屏幕输出任何内容。因此，要想使存储过程向屏幕输出内容，必须打开 SERVEROUTPUT，即执行 SET SERVEROUTPUT ON 命令。

STEP 01 创建存储过程 p_sub1。

```
CREATE OR REPLACE PROCEDURE p_sub1
is
BEGIN
 dbms_output.put_line('data FROM subprograme!');
END;
```

存储过程 p_sub1 的功能是在屏幕上输出一段内容。

STEP 02 执行存储过程。

```
SQL> EXECUTE p_sub1;
```

执行存储过程，没有结果输出。为什么呢？这是因为 SERVEROUTPUT 没有打开。

STEP 03 打开 SERVEROUTPUT。

```
SQL> SET SERVEROUTPUT ON;
```

STEP 04 重新执行存储过程。

5．存储过程不能访问表的问题

角色在函数、过程、触发器中都是失效的，也就是说，用户从角色中继承过来的权限，不能在函数、过程、触发器中使用。在函数、过程、触发器中，如果要访问其他用户的对象，需要显式地给用户授予访问的权限。

接下来，我们将通过一个例子演示从角色中继承来的权限是如何在函数、过程、触发器中使用的。假如用户 wy 有一表 t_proc，该表上的 SELECT 权限被授予角色 rol_sel（GRANT SELECT ON wy.t_proc TO rol_sel），而角色 rol_sel 被授予用户 acc。

STEP 01 以用户 acc 的身份登录数据库。

STEP 02 创建存储过程 p_proc，存储过程将访问用户 wy 的表 t_proc。

```
CREATE OR REPLACE PROCEDURE p_proc IS
  v_a INTEGER;
BEGIN
  SELECT MAX(a) INTO v_a FROM wy.t_proc;
END;
```

STEP 03 存储过程创建失败，显示错误信息。

```
SQL> SHOW ERROR;
-------- ---------------------------------
```

```
4/34    PL/SQL: ORA-00942: 表或视图不存在
```

从提示中我们可以知道，用户 acc 不能访问用户 wy 的表 t_proc（即使角色 rol_sel 有访问该表的权限）。这需要我们把表 t_proc 上的 SELECT 权限直接授予用户 acc。

STEP 04 把表 t_proc 上的 SELECT 权限直接授予用户 acc

```
GRANT SELECT ON wy.t_proc to acc;
```

STEP 05 重新运行创建存储过程 p_proc 的脚本，存储过程 p_proc 创建成功。

6．加密存储过程（函数、包）

Oracle 提供了一个叫 wrap 的操作系统命令行工具，用来加密存储过程（Procedure）、函数（Function）、包（Package）等 PL/SQL 源代码。wrap 加密存储过程、函数、包。防止其他用户对用户商业逻辑的偷窥，帮助保守商业密码。

wrap 的语法如下所示：

```
wrap iname=input_file [oname=output_file]
```

其中，input_file 是输入文件的名字，该文件中含有源代码，output_file 是输出文件的名字。

示例：wrap 加密，本例将对存储过程 wraptest 进行加密。

STEP 01 创建文件 wrap_test.sql，这个文件的内容如下所示：

```
CREATE PROCEDURE wraptest IS
BEGIN
dbms_output.put_line('wrap successfully!');
END;
```

这个文件包含创建存储过程 wraptest 的脚本。

STEP 02 使用 wrap 对存储过程进行加密。

```
C:\Documents and Settings\Administrator>wrap iname=D:\temp\wrap_test.sql
PL/SQL Wrapper: Release 11.1.0.6.0- Production on 星期一 10 月 12 17:24:07
2009
Copyright (c) 1993, 2004, Oracle.  All rights reserved.
Processing D:\temp\wrap_test.sql to wrap_test.plb
```


其中，iname 指定输入文件，输入文件（wrap_test.sql）是在创建的文件，其含有创建存储过程 wraptest 的脚本。wrap 命令会输出（生成）一个新的文件，这个输出文件可以使用参数 oname 指定其路径及名字；如果没有指定参数 oname，wrap 在当前路径下生成和输入文件名字相同但是后缀（.plb）不同的文件。在本例中，wrap 自动生成文件 wrap_test.plb。

STEP 03 运行 wrap 生成的文件 wrap_test.plb。

```
SQL> @D:\temp\wrap_test.plb;
```

这个脚本将创建存储过程 wraptest。生成的 wrap_test.plb 文件位于当前目录，这一点需要留意。

STEP 04 打开 serveroutput。

```
SQL> set serveroutput on;
```

STEP 05 执行存储过程 wraptest。

```
SQL> execute wraptest;
wrap successfully!
PL/SQL procedure successfully completed.
```

从输出结果可以看出，存储过程成功执行。

STEP 06 查看存储过程的源代码。

```
SELECT name,text
FROM all_source
WHERE type='PROCEDURE' and name='WRAPTEST';
```

SQL 语句的执行结果是：

```
TEXT
------------------------------------------------------------------------
abcd
abcd
abcd
…
Sv4osr3nsrMdBjAsriTqsjLIewIGJV1ddwYGoCVubvLpMi720eokH/Y5pk1OHwM=
```

从输出可以看到，我们已经无法看到存储过程 WRAPTEST 的内容（源代码）。

特别要注意的是，如果对存储过程进行加密，一定要记得保留创建存储过程的源代码。

7．用存储过程返回一个结果集

在 20.2.4 节中，我们学习了游标变量（REF CURSOR），游标变量可以作为存储过程和函数的参数，利用游标变量的这个特性，我们可以在存储过程中返回 SELECT 语句的结果集。利用游标变量返回结果集示例如下所示：

STEP 01 定义包头。

```
CREATE OR REPLACE PACKAGE pack_return_result AS
  TYPE type_cursor IS REF CURSOR;--定义游标指针，这里定义了一种新的类型
-- p 是游标变量，一定要把参数的模式设置成 OUT，存储过程只能用这种方式往外传递数据。
  PROCEDURE pro_getresult(p IN OUT type_cursor);
 END pack_return_result;
```

STEP 02 定义包体。

```
CREATE OR REPLACE PACKAGE BODY pack_return_result AS
  PROCEDURE pro_getresult(p IN OUT type_cursor) AS
  BEGIN
    OPEN p FOR
      SELECT phone_name, producer, price
```

```
        FROM phone_infor
       WHERE price > 500;
  END;
END;
```

STEP 03 打开 SERVEROUTPUT。

```
SQL> SET SERVEROUTPUT ON
```

STEP 04 定义游标绑定变量。

```
SQL> VARIABLE v_bind_ref REFCURSOR
```

其中，VARIABLE 不是 SQL 语句，是 SQL*Plus 命令。

知识点索引——游标绑定变量（REFCURSOR Bind Variable）

在 20.1.3 节中，我们学习了绑定变量，在这里，我们要学习游标绑定变量（REFCURSOR Bind Variable），游标绑定变量是特殊类型的绑定变量，用于批量处理 PL/SQL 中 SELECT 语句的结果。使用 SQL*Plus 命令 PRINT，可以打印游标绑定变量的内容。

STEP 05 执行包（pack_return_result）中的存储过程（pro_getresult）。

```
SQL> EXECUTE pack_return_result.pro_getresult(:v_bind_ref)
```

STEP 06 打印游标绑定变量的内容。

```
SQL> PRINT v_bind_ref
```

SQL 语句的执行结果是：

```
PHONE_NAME          PRODUCER                PRICE
--------------------------------------------------
N97                  诺基亚                   700
C510c                爱立信                   1201
……
```


8．从数据库中获得创建存储过程的脚本

视图 USER_SOURCE/ DBA_SOURCE 用于存放模式对象创建的脚本，查询这个视图，可以得到存储过程创建的脚本。

```
SQL> SELECT text
  2    FROM user_source
  3   WHERE NAME = 'PRO1'
  4     AND TYPE = 'PROCEDURE';
```

SQL 语句的执行结果是：

```
TEXT
```

```
-------------------------
PROCEDURE pro1
AS
v1 NUMBER;
BEGIN
NULL;
END;
```

其中，PRO1 是存储过程的名字，TYPE 用于指定对象类型，我们需要的是存储过程的源代码，因此 TYPE = 'PROCEDURE'。

这里得到的是一段不完整的脚本，不能直接运行这段脚本重建存储过程 PRO1。如果在上面的输出前面加上 CREATE OR REPLACE，就可以利用该脚本重建存储过程 PRO1。



20.4 函数（Function）

函数也是编译好且存储在数据库服务器上的程序代码。函数有名字及自己的参数。与存储过程不同的是，函数必须**有返回值**。函数具有和存储过程相同的优点。Oracle 内置了很多函数，我们也可以开发自己的函数。



20.4.1 函数的创建

如果只想在自己的模式中创建函数，则只需具有 CREATE PROCEDURE 权限；如果想在其他模式下创建函数，用户必须具有 CREATE ANY PROCEDURE 权限。拥有 ALTER ANY PROCEDURE 权限的用户可以替换其他模式的函数。

与匿名块、存储过程一样，函数也是由 3 部分组成：定义部分、处理部分、异常处理部分。函数定义的语法如下所示：

```
CREATE [ OR REPLACE] FINCTION [schema.]function_name [parameter_list]
RETURN returning_datatype
{AS|IS}
declaration_section
BEGIN executable_section
[EXCEPTION]
exception_section
END [function_name]
```

其中，OR REPLACE 表示如果函数已经存在，则替换已经存在的函数。schema 表示模式名，将在指定的模式下创建函数。function_name 表示函数的名字。parameter_list 指定函数的参数，定义函数时，要指定函数的数据类型（也可以使用%TYPE 或者%ROWTYPE 指定参数的数据类型）、默认值（使用 DEFAULT 指定）、参数模式。有三种参数模式：IN 表示调用函数时，必须给参数指定一个值，参数的模式默认是 IN；OUT 表示参数可以在函数内被修改；IN OUT 是前面两种的组合。returning_datatype 定义函数返回值的数据

类型，数据类型可以是标量类型，也可以是复合类型。定义函数返回值的数据类型是必需的。declaration_section 函数的定义部分，主要用于定义变量，这部分可选。executable_section 执行部分，含有一系列的 PL/SQL 语句，这部分必选。exception_section 异常处理部分，用于异常处理，这部分可选。END 表示函数定义的结束。

示例：函数创建。

```
CREATE OR REPLACE FUNCTION F2(par1 IN NUMBER DEFAULT 500)
RETURN NUMBER
IS
  V1 NUMBER;
BEGIN
  SELECT 1
  INTO v1
  FROM dual;
  RETURN(V1);
END;
```

其中，F2 是函数的名字；par1 是函数的参数，数据类型是 NUMBER，参数的模式是 IN，默认值是 500；RETURN NUMBER 表示把函数返回值的数据类型定义成 NUMBER；RETURN(V1)表示把变量 V1 的值作为函数的返回值，特别注意的是，函数一定要有返回值。



20.4.2 函数的调用

在很多地方都可以调用函数：在 SQL 语句中调用函数、在赋值语句中调用函数、在表达式中调用函数……

■ 在 SQL 语句中调用函数

```
SELECT f1(1) FROM DUAL;
```

其中，f1 是函数的名字，1 是函数的参数的传入值。

■ 在赋值语句中调用函数

```
a:= f1(1);
```

其中，a 是变量，f1 是函数的名字。

■ 在表达式中调用函数

```
IF f1(1)+7 > 10000
THEN
```

在这里，函数是表达式的一部分。

■ 没有参数的函数的调用

如果一个函数没有参数，则调用该函数时，不需要括号。

```
SELECT F3 FROM DUAL;
```

■ 函数定义了参数，并且没有指定参数的默认值，则在调用时，如果不指定参数值，

会发生错误

```
SQL> SELECT F4 FROM DUAL;
ORA-06553: PLS-306: 调用 'F4' 时参数个数或类型错误
```



20.4.3 函数的编译

如果函数变得无效，需要对函数进行编译。使用命令 ALTER FUNCTION…COMPILE 对函数进行编译。

```
ALTER FUNCTION f4 COMPILE;
```

其中，f4 是函数以的名字。



20.4.4 函数的删除

使用 DROP FUNCTION 命令删除函数。如果一个函数是包的一部分，则不能使用该命令删除包中的函数。要想删除其他用户创建的函数，执行删除的用户必须被授予 DROP ANY PROCEDURE 权限。

示例：删除函数 F4。

```
SQL> DROP FUNCTION f1;
```



20.4.5 函数使用技巧

本节将展示函数使用中的一些技巧。

1．用函数返回一个结果集

利用函数返回一个结果集的原理和利用存储过程返回一个集合的原理相同。利用存储过程返回一个集合的例子**请参考本书 20.3.5 节的内容**。利用函数返回一个结果集的例子说明如下。



定义函数 fun_getresult。

```
CREATE OR REPLACE FUNCTION fun_getresult
  RETURN pack_return_result.type_cursor IS-- 类型 type_cursor 来自于“21.3.5.7 用存储过程返回一个结果集”定义的包 pack_return_result 中。
  p pack_return_result.type_cursor;--定义游标变量
BEGIN
  OPEN p FOR--打开游标
    SELECT phone_name, producer, price FROM phone_infor WHERE price > 500;
  RETURN(p);--返回游标变量
END;
```



打开 SERVEROUTPUT。

```
SQL> SET SERVEROUTPUT ON
```

STEP 03 定义游标绑定变量。

```
SQL> VARIABLE v_bind_ref REFCURSOR
```

STEP 04 调用函数 fun_getresult，函数返回值存放在绑定游标变量 v_bind_ref 中。

```
EXECUTE :v_bind_ref := fun_getresult();
```

STEP 05 打印游标绑定变量的内容。

```
SQL> PRINT v_bind_ref
```

2．在函数中执行 DDL

在函数中也可以执行 DDL 语句。在函数中执行 DDL 语句，和在存储过程中执行 DDL 语句的原理相同，都是利用动态 SQL（EXECUTE IMMEDIATE 或者 DBMS_SQL 包）。利用本地动态 SQL（EXECUTE IMMEDIATE）动态创建表的例子如下。

STEP 01 创建函数 F4，函数将根据不同的名字创建不同的表。

```
CREATE OR REPLACE FUNCTION F4(tablename IN VARCHAR2)
RETURN NUMBER
IS
PRAGMA AUTONOMOUS_TRANSACTION;
  V1 NUMBER;
BEGIN
EXECUTE IMMEDIATE  'CREATE TABLE ' || tablename || ' ( name VARCHAR(20))';
  RETURN(V1);
END;
```

STEP 02 运行函数 F4。

```
DECLARE
  v_r NUMBER; --存放函数返回值
BEGIN
  v_r := f4('bbcc');--执行函数调用
END;
```


STEP 03 验证表是否创建成功。

```
SQL> DESC bbcc
Name Type        Nullable Default Comments
---- ------------ -------- ------- --------
NAME VARCHAR2(20) Y
```

3．函数中执行 DML

DML 在函数中的执行和在存储过程中的执行相同，没有太大区别。下面是在函数中执行 DML 语句的示例。

STEP 01 创建函数 f_phone。

```
CREATE OR REPLACE FUNCTION f_phone RETURN VARCHAR2 IS
BEGIN
  INSERT INTO phone_infor VALUES ('3310', '诺基亚', 111);
  COMMIT;
  RETURN 'ok';
END;
```



在块中调用函数 f_phone。

```
DECLARE
  v_r VARCHAR2(200);
BEGIN
  v_r := f_phone;
  dbms_output.put_line(v_r);
END;
```

4．错误 ORA-14552 和 ORA-14551 的解决

如果在 SELECT 语句中调用含有 DDL 或者是 DML 的函数，将会产生错误："ORA-14552: 在查询或 DML 中无法执行 DDL，提交或回退"或者"ORA-14551: cannot perform a DML operation inside a query"。通常情况下，不允许在查询语句中使用含有 DDL 或者 DML 语句的函数。但是，若还是想使用它们，必须利用**自治事务**（Autonomous Transactions，AT）原理来解决。自治事务是由另外一个事务调用的，调用它的事物叫主事务（Main Transaction，MT）。自治事务的提交、回滚并不影响主事务（有关自治事务的原理，**请参考 16.2.6 节的内容**）。



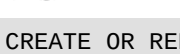
创建函数 f5，该函数的功能是动态创建表。

```
CREATE OR REPLACE FUNCTION f5(tablename IN VARCHAR2) RETURN NUMBER IS
  v1 NUMBER := 88;
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE ' || tablename || ' ( name VARCHAR(20))';
  RETURN(v1);
END;
```



在 SELECT 语句中调用函数 f5，发生错误。

```
SQL> SELECT f5('tab_f5') FROM dual;
ORA-14552: 在查询或 DML 中无法执行 DDL, 提交或回退
```



将创建函数 f5 的 SQL 语句改写成如下形式：

```
CREATE OR REPLACE FUNCTION f5(tablename IN VARCHAR2) RETURN NUMBER IS
  PRAGMA AUTONOMOUS_TRANSACTION;--使用事务自治
  v1 NUMBER := 88;
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE ' || tablename || ' ( name VARCHAR(20))';
  RETURN(v1);
```

```
END;
```

STEP 04 重新执行查询语句，执行成功。

```
SQL> SELECT f5('tab_f5') FROM dual;
```

STEP 05 验证表 tab_f5 是否创建成功。

```
SQL> DESC tab_f5
```

5．从数据库中获得创建函数的脚本

从数据库中获得创建函数的脚本和 21.3.5 节的原理一样，都是查询数据字典获得创建函数的脚本。

示例：获得创建函数 F1 的脚本。

```
SELECT text FROM user_source WHERE name='F1' and type='FUNCTION';
```

20.4.6 常用函数

本节将节选部分常用的函数进行介绍。

1．常用的数字型函数

常用数字型函数的用法如表 20-1 所示。

表 20-1 常用数字型函数

函　　数	用法说明
ABS(n)	返回数字 n 的绝对值
ACOS(n)	返回 n 的反余弦值
CEIL(n)	返回大于或等于 n 的最小的整数值
COS(n)	返回 n 的余弦值
EXP (n)	返回 e 的 n 次幂
FLOOR(n)	返回小于等于 n 的最大整数值
HEXTORAW(c)	将包含十六进制的 c 转换为一个 RAW 数值
LN (n)	返回 n 的自然对数
LOG (m, n)	返回以 m 为底 n 的对数
MOD (m, n)	返回 m 除以 n 的余数，如果 n=0 则返回 m 的值
POWER (m, n)	返回 m 的 n 次方根
RAWTOHEX	将 RAW 数值转换为一个包含十六进制的字符值
ROUND (m, [n])	四舍五入，返回小数点右边 n 位后 m 的值，n 默认值为 0
ROWIDTOCHAR	将 ROWID 类型转换成字符型
SIGN(n)	判断数的符号，如果 n<0 返回-1；如果 n>0 返回 1；如果 n=0 返回 0

SIN(n)	返回 n 的正弦值，n 为弧度

续表

函　　数	用法说明
SQRT(n)	返回 n 的平方根，n 为弧度，n□0
TAN(n)	返回 n 的正切值，n 为弧度
TO_CHAR(n)	将数字转换成字符
TO_NUMBER(c)	将字符转换成数字
TRUNC (m, [n])	对数字 m 按照指定的位数 n 进行截断，n 为负数表示从小数点左边开始截断，为正表示从小数点右边开始截断
UID	返回一个唯一标识当前数据库用户的整数

2．常用的字符型函数

常用字符型函数的用法如表 20-2 所示。

表 20-2　常用字符型函数

函　　数	用法说明
ASCII(c)	返回给定字符的 ASCII 码
CHARTOROWID(c)	将字符串转换为 ROWID 类型
CHR(n)	返回数字 n 对应的字符
CONCAT(c1,c2)	连接两个字符串 c1 和 c2
CONVERT(c,DEST_CHAR_SET,SOURCE_CHAR_SET)	将字符串 c 中的字符从字符集 SOURCE_CHAR_SET 表示的字符转换为 DEST_CHAR_SET 表示的字符
INITCAP(c)	将字符串中单词的第一个字母转换为大写，其他字母则转换为小写
LENGTH（c）	返回指定字符串的长度
LOWER（c）	将给定字符串转换成小写
LPAD(c1,n[,c2])	创造一个以字符串 c1 结尾、总长度是 n 的字符串，c1 前面没有的字符用字符串 c2 进行填充。如果没有指定 c2，则用空格填充
LTRIM(c1[,c2])	从字符串 c1 左侧开始，去掉与字符串 c2 相同的字符。如果 c2 为空则去掉空格
RPAD(c1,n[,c2])	功能和 LPAD 相同，只是方向相反
RTRIM(c1[,c2])	与 LTRIM 用法相同，只是方向相反
UPPER	将给定字符串转换成大写
USERENV	返回当前会话的信息
NVL(c1,c2)	如果 c1 为 NULL 则返回 c2，否则返回 c1

3．常用的日期型函数

常用日期型函数的用法如表 20-3 所示。

表 20-3　常用日期型函数

函　　数	用法说明
ADD_MONTHS(n)	返回指定日期 n 个月之后的日期
LAST_DAY(d)	返回指定日期所在月的最后一天
MONTHS_BETWEEN(d1,d2)	返回两个日期之间的月份差
NEXT_DAY(d,w)	返回指定日期（d）之后星期几（w）的日期
ROUD(DATE,FORMAT)	对日期按照指定的格式进行四舍五入
SYSDATE	返回系统的当前日期和时间
SYSTEMSTAMP	返回当前系统的时间戳
TO_CHAR	将日期型转换成字符型
TO_DATE	将字符型转换成日期型
TRUNC(DATE,FORMAT)	返回 date 经过截断后的日期，截断的位置由 FORMAT 指定。如果省略 FORMAT，则默认为截断到天



20.5　包（Package）

包也是模式对象，它是将相关对象（元素）集成在一起的 PL/SQL 结构。包由两部分组成：包头（Package Specification）和包体（Package Body），每个部分都被单独存放。与函数、存储过程一样，包被存放在数据库的数据字典中。可以被包集成的对象有：

- 存储过程（Procedures）
- 函数（Functions）
- 游标（Cursors）
- 变量（Variables）
- 常量（Constants）
- 异常（Exception）
- PL/SQL 表（PL/SQL Table）
- 记录类型（Record TYPE）



20.5.1　包的优点

如果需要多个函数或者过程协同工作完成一个任务，这时候，我们可以考虑把这些函数和存储过程集成在一个包中，这样方便函数和存储过程的管理。包有下面的优点：

- 隐藏商业规则，提高数据安全性。
- 提高程序执行的性能，当第一次调用包时，包被整体读入内存，后续的调用不会耗

费额外的 I/O，当然，还有其他特性也会有助于提高性能。

- 加强程序开发的模块化。
- 能够为用户提供全局的信息。
- 方便利用面向对象的设计思想（Object-Oriented Design）进行程序设计。



20.5.2 包的组成

包由包头（Package Specification）与包体（Package Body）两部分组成。“包头”可以叫“包的定义部分”，也可以叫“包的说明部分”；包体可以叫“包的主体”。

1．包头（Package Specification）——包的说明部分

包头定义包的调用接口，列出了包中包含的对象。外部程序可以访问（调用）这些对象。在包头，可以定义变量、常量、类型（TYPE）、游标、异常、存储过程和函数。包头定义的例子，**请参考 20.5.4 节的内容**。

2．包体（Package Body）——包的主体

包头只是定义包中对象（如存储过程和函数）的调用接口，实现这些对象的代码被放在包体。简单地说，包头只是包的一个空外壳，真正的功能实现在包体完成。

在包体中可以引用包头定义的对象（变量、常量、函数等）。在包体中也可以定义自己的变量、常量、函数等对象，但是，这些对象只在包体中有效，只能被包中的其他子程序所调用。

如果在包头部分定义了游标，则在包体要给游标指定 SELECT 语句。如果包头定义了存储过程或者函数，则要在包体编写实现这些存储过程或者函数的代码。



20.5.3 包中对象的私有属性和公有属性

在包头定义的对象，包外面的程序可以对其进行引用，当然，包内的存储过程、函数等也可以对其进行引用；在包体中定义的对象（如函数等）只能在包体内进行引用，包外的对象不能对它们进行引用。



20.5.4 包的创建

创建包的过程分成两个部分：第一部分创建包的定义（包头）；第二部分创建包体。要想在自己的模式下创建包，用户必须具有 CREATE PROCEDURE 权限；如果想在其他模式下创建包，用户必须要有 CREATE ANY PROCEDURE 权限。

■ 创建包头的语法如下所示：

```
CREATE [OR REPLACE] PACKAGE package_name
{AS|IS}
```

```
    public_variable_declarations |
    public_type_declarations |
    public_exception_declarations |
    public_cursor_declarations |
    function_declarations |
    procedure_declarations
    END [package_name]
```

其中，OR REPLACE 表示替换已经存在的包头。package_name 表示包的名字。public_variable_declarations 定义全局变量。public_type_declarations 为用户自定义数据类型（全局的）。public_exception_declarations 定义全局异常。public_cursor_declarations 定义全局游标。function_declarations 表示包中函数的接口定义。procedure_declarations 表示包中存储过程的接口定义。

示例：创建包头。

```
    CREATE OR REPLACE PACKAGE pack_first AS
      b VARCHAR2(10);--定义变量
      c_id CONSTANT NUMBER := 50;--定义常量
      TYPE emprectyp IS RECORD(
        emp_id INT,
        salary REAL);--定义类型
      CURSOR caller_cur RETURN emprectyp;--定义游标变量
      invalid_salary EXCEPTION;--定义异常
      FUNCTION hire_employee(comm IN REAL, deptno IN REAL) RETURN INT;--定义
函数
      PROCEDURE fire_employee(emp_id INT, v3 IN VARCHAR2);--定义存储过程
    END pack_first;
```

这里只定义包的调用接口。

■ 创建包体的语法如下所示：

```
    CREATE [OR REPLACE] Package Body package_name
    {AS|IS}
    private_variable_declarations |
    private_type_declarations |
    private_exception_declarations |
    private_cursor_declarations |
    function_declarations |
    procedure_declarations
    BEGIN
    initialization part
    END [package_name]
```

其中，OR REPLACE 表示覆盖已经存在的包体。package_name 表示包的名字，这个名字必须和包定义部分的名字相同。private_variable_declarations 定义私有变量。private_type_declarations 定义私有数据类型。private_exception_declarations 定义私有异常。

private_cursor_declarations 定义私有游标。function_declarations 定义函数，这里真正实现函数的功能。procedure_declarations 定义存储过程，这里真正实现存储过程的功能。BEGIN 到 END 之间的部分叫包的初始化部分，用于变量初始化，变量初始化发生在第一次引用包中对象（元素）时。

示例：创建包体。

```
CREATE OR REPLACE PACKAGE BODY pack_first AS
CURSOR caller_cur RETURN emprectyp
   IS  SELECT emp_id,salary FROM employee;--给游标指定 SELECT 语句
  number_hired INT;--定义包体的变量，该变量只在包体内有效
  v_getfun INT;

  --定义函数
  FUNCTION hire_employee(comm IN REAL, deptno IN REAL) RETURN INT IS
    new_empno INT;
  BEGIN
    b:='rxxr';--在函数中引用包头中定义的变量
    new_empno := 1;
    RETURN new_empno;
  END hire_employee;

--定义存储过程
  PROCEDURE fire_employee(emp_id INT, v3 IN VARCHAR2) IS
  BEGIN
    NULL;
    number_hired:=44;--引用包体中定义的变量
    v_getfun:=hire_employee(11,number_hired);--调用包头定义的函数
    dbms_output.put_line('I am from a PACKAGE');
  END fire_employee;

-- 初始化部分
BEGIN
  b:= 1;--对变量进行初始化
END pack_first;
```




20.5.5 引用包的对象（元素）

在包外，我们可以引用包中的存储过程、函数、类型、变量、常量。引用格式是：包名.对象名（包名和对象名之间是圆点）。

■ 调用包中的存储过程



打开 SERVEROUTPUT。

```
SQL>SET SERVEROUTPUT ON
```



调用包 pack_first 中的存储过程 fire_employee。

```
SQL>EXECUT pack_first.fire_employee(11,22);
I am from a PACKAGE
PL/SQL 过程已成功完成。
```

■ 调用包 pack_first 中的函数 hire_employee，引用包中的类型和常量

```
SQL> DECLARE
  2    a INT;
  3    vv_cur pack_first.emprectyp;--引用包 pack_first 中的类型 emprectyp
  4  BEGIN
  5 a:=pack_first.hire_employee(10, 20);--调用包 pack_first 的函数 hire_empl
oyee
  6    dbms_output.put_line(a);
  7    dbms_output.put_line(pack_first.c_id);--引用包 pack_first 中的常量 c_id
  8  END;
  9  /
1
50
PL/SQL 过程已成功完成。
```

20.5.6 让其他用户使用你的包

包是一个对象，只能整体授权，不能把包中一个对象的使用权限单独授予某个用户。

■ 把包 pack_first 的执行权限授予用户 wy

```
SQL> GRANT EXECUTE ON pack_first TO wy;
```

授权成功。

■ 把包 pack_first 中存储过程 fire_employee 的执行权限授予用户 wy（错误）

```
SQL> GRANT EXECUTE ON pack_first.fire_employee TO wy;
ORA-04042: 过程, 函数, 程序包或程序包体不存在
```

20.5.7 包的编译

如果包变得无效，可以对包重新进行编译。

■ 对包进行编译（包括包头和包体）

```
ALTER PACKAGE pack_first COMPILE PACKAGE;
```

■ 对包头进行编译

```
ALTER PACKAGE pack_first COMPILE SPECIFICATION;
```

■ 对包体进行编译

```
ALTER PACKAGE pack_first COMPILE BODY;
```

20.5.8 包的删除

使用 DROP PACKAGE 命令删除包，要想删除其他用户创建的包，则执行删除的用户必须拥有 DROP ANY PROCEDURE 权限。不能单独删除包中的函数或者存储过程，如果真想删除它们，可以重建包，在重建的时候把它们过滤掉。

■ 只删除包体，包头仍然存在

```
SQL> DROP PACKAGE BODY pack_first;
```

其中，pack_first 是包的名字。

■ 删除整个包（包括包头和包体）

```
SQL> DROP PACKAGE pack_first;
```

20.5.9 包中存储过程和函数的重载

重载就是创建多个名字相同的存储过程或者函数，但是，它们的参数个数和参数类型不同。当调用这样的函数或者存储过程时，即使存储过程或者函数的名字相同，根据不同的参数个数和参数类型，Oracle 会自动调用相应的存储过程或者函数。

示例：存储过程重载。

下面将以重载存储过程 f_get_infor 为例，无论用户输入的是学生的姓名（name）还是学生的编号（ID），存储过程 f_get_infor 都可以返回学生的信息。

STEP 01 学生信息表如下所示。

```
SQL> DESC students3
```

SQL 语句的执行结果是：

```
 Name                NULL?   Type
 ------------- ------- ---------------
 ID                          NUMBER(38)
 NAME                    VARCHAR2(20)
 CITY                    VARCHAR2(20)
```

其中，ID 表示学生的编号；NAME 表示学生的姓名；CITY 表示学生所在的城市。

STEP 02 定义包头。

```
SQL> CREATE OR REPLACE PACKAGE pkg_student AS
  2    PROCEDURE f_get_infor(p_StudentID  IN students3.id%TYPE);
  3    PROCEDURE f_get_infor(p_name IN students3.name%TYPE)
;
  4  END pkg_student;
  5  /
Package created.
```

在包 pkg_student 中，对存储过程 f_get_infor 进行了重载。虽然这两个存储过程的名

字相同，但是，它们的参数不同（一个是 NUMBER 型，一个是 VARCHAR2 型）。

STEP 03 定义包体。

```
SQL> CREATE OR REPLACE Package Body pkg_student AS
--输入参数 p_StudentID 是学生的 ID
  2   PROCEDURE f_get_infor(p_StudentID  IN students3.id%TYPE)
  3   IS
  4   v_id int;
  5   v_name VARCHAR2(200);
  6   v_city VARCHAR2(20);
  7   BEGIN
  8   SELECT id,name,city INTO v_id,v_name,v_city
  9   FROM students3
 10   WHERE id=p_StudentID;
 11     dbms_output.put_line('ID:  '||v_id||'  NAME:  '||v_name||'  city:
'||v_city);
 12   END f_get_infor;
 13
--输入参数 p_name 表示学生的姓名
 14   PROCEDURE f_get_infor(p_name  IN students3.name%TYPE)
 15   IS
 16   v_id int;
 17   v_name VARCHAR2(200);
 18   v_city VARCHAR2(20);
 19   BEGIN
 20   SELECT id,name,city INTO v_id,v_name,v_city
 21   FROM students3
 22   WHERE name=p_name;
 23     dbms_output.put_line('ID:  '||v_id||'  NAME:  '||v_name||'  city:
'||v_city);
 24   END f_get_infor;
 25  END pkg_student;
 26  /
Package body created.
```

这里定义了两个存储过程 f_get_infor，一个存储过程的输入参数是学生的编号（ID），一个存储过程的输入参数是学生的姓名（NAME），两个存储过程都会得到学生的信息。

STEP 04 打开屏幕输出。

```
SQL> SET SERVEROUTPUT ON;
```

STEP 05 查看学生的信息。

```
SQL> SELECT * FROM students3;
```

SQL 语句的执行结果是：

```
       ID   NAME        CITY
---------- ------------   -------------------------
       12   KATE        HANGZHOU
```

STEP 06 执行存储过程 f_get_infor，传入参数是学生的编号（ID）12。

```
SQL> EXECUTE pkg_student.f_get_infor(12);
ID: 12 NAME: KATE city: HANGZHOU
PL/SQL procedure successfully completed.
```

存储过程 f_get_infor 正确输出了学生的信息。

STEP 07 执行存储过程 f_get_infor，传入参数是学生的名字（NAME）KATE。

```
SQL> EXECUTE pkg_student.f_get_infor('KATE');
ID: 12 NAME: KATE city: HANGZHOU
PL/SQL procedure successfully completed.
```

存储过程 f_get_infor 正确输出了学生的信息。

从上面可以看出，无论输入的是学生的编号还是学生的姓名，存储过程 f_get_infor 都可以得到学生的信息。



注意：存储过程（函数）重载的注意事项

- **如果两个函数（存储过程）仅仅是名字上或者参数的模式上不同，那么，它们不能算是重载。**

```
PROCEDURE TBB(P IN INT)
PROCEDURE TBB(P OUT INT)
```

上述两个参数模式，一个是 IN，一个是 OUT，仅仅是参数模式上不同，这样的存储过程不能算重载。

- **返回值的类型不同也不能算重载。**

```
FUNCTION F1 RETURN DATE
FUNCTION F1 RETURN NUMBER
```

一个函数的返回值是 DATE 型，一个函数的返回值是 NUMBER 型，返回值的类型不同，也不能算重载。

- **重载函数的参数类型不能属于同一族。**

```
PROCEDURE PS(P IN CHAR)
PROCEDURE PS(P IN VARCHAR2)
```

一个存储过程的参数是 CHAR，一个函数的参数是 VARCHAR2，这样也不能算重载，因为 CHAR 和 VARCHAR2 同属于一族（字符型）。



20.5.10 Oracle 内置包

用户可以开发自己的包，也可以使用 Oracle 提供的包，我们把 Oracle 提供的包叫“Oracle 内置包”。Oracle 内置包有 DBMS_ALERT、DBMS_OUTPUT、DBMS_PIPE、DBMS_CONNECTION_POOL、HTF 、HTP、UTL_FILE、UTL_HTTP、UTL_SMTP 等。这些包功能强大，每个包针对某一方面的应用（如 UTL_SMTP 包用于发送电子邮件）。

20.5.11 查看包中的存储过程或者函数

使用 DESC 命令或者查询字典视图 DBA_PROCEDURES，均可知道一个包中有哪些存储过程和函数。

本文以包 PACK_FIRST 为例。

■ 查询字典视图

```
SQL> SELECT object_name package_name,procedure_name
  2  FROM dba_procedures
  3  WHERE owner='ITME' and object_name='PACK_FIRST'
```

其中，PACK_FIRST 是包的名字；ITME 是包 PACK_FIRST 的拥有者。

■ 使用 DESC 命令

```
SQL> DESC PACK_FIRST
```

这种方法比较直观，可以区分包中拥有的是存储过程还是函数。

20.6 存储过程和函数的调试（Debug）

有多种方法可以调试 Oracle 的函数和存储过程，根据实际开发的需要，建议大家使用 PL/SQL Developer 进行函数和存储过程的调试。PL/SQL Developer 提供图像界面，调试过程简单、方便。

本文将以调试函数为例，讲述如何使用 PL/SQL Developer 调试函数 rmb_daxie。函数 rmb_daxie 的定义如下：

```
CREATE OR REPLACE FUNCTION rmb_daxie(p_suzi NUMBER) RETURN VARCHAR2
...
```

其中，p_suzi 为函数的输入参数。

STEP 01 启动 PL/SQL Developer，登录数据库，找到函数 rmb_daxie，在函数上单击鼠标右键，弹出菜单，在菜单上选择“Test”，进入调试界面。

STEP 02 在如图 20-1 所示的界面中输入函数的传入参数。

	Variable	Type	Value
▶	result	String	
	p_suzi	Float	5

图 20-1 设置传入参数的值

输入函数 rmb_daxie 的传入参数 p_suzi 的值为 5；result 的值不需要输入，因为 result 表示函数的返回值。

单击 按钮或者按 F9 键开始调试函数。

STEP 03 断点设置。

把光标移动到要设置断点的行，按 Ctrl+B 组合键或者用鼠标单击该行起始边缘，效果如图 20-2 所示。

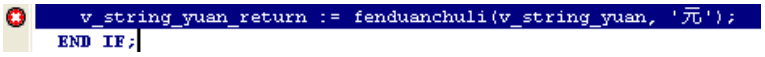


图 20-2　设置断点

STEP 04 跟踪变量值的变化。

如果想跟踪某个变量值的变化，在如图 20-3 所示的窗口中输入变量的名字（本例中是 v_string_yi_return），调试的过程中，该窗口将不断显示变量（v_string_yi_return）的值的变化。

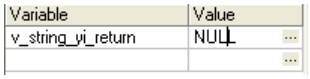


图 20-3　显示变量值

STEP 05 开始调试。

调试时，有几种方法可供选择：

表示运行到结束。

表示进入被调用的存储过程或者函数。

表示在执行中遇到函数（存储过程）调用时，跳过被调用者。

表示从程序单元（函数、存储过程）中跳出。

表示执行到下一个断点。

STEP 06 调试完毕，结果如图 20-4 所示。

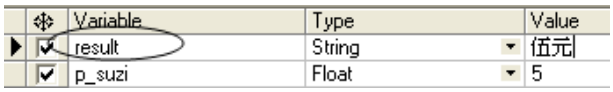


图 20-4　函数返回值

调试完成后，你将会看到函数的返回值。本例中，函数 rmb_daxie 的返回值是“伍元”。



本章总结

本章主要介绍存储过程、函数和包。除了介绍它们的语法之外，还展示了存储过程、函数和包使用的一些技巧（很多技巧都来源于开发）。我们都知道，存储过程和函数难于跟踪调试，本书给出了一种方法，让你可以对它们进行调试。学完本章，开发部分的基础知识我们也介绍得差不多了，希望大家多多练习。

第 21 章　Oracle 即时客户端（Instant Client）

Oracle 即时客户端（Instant Client）是一个很值得推荐的东西，它让你的应用程序部署变得非常容易，不信就试试看吧！



21.1　即时客户端简介

Oracle 即时客户端允许在没有安装 Oracle 客户端（或者没有 ORACLE_HOME）的情况下运行你的应用程序。无须改变你的应用程序，OCI、OCCI、Pro*C、ODBC、JDBC 等应用程序便可工作。甚至在没有安装 Oracle 客户端的情况下，你也可以使用 SQL*Plus。Oracle 即时客户端帮我们节省了磁盘空间，简化了应用程序的部署。

Oracle 即时客户端只需要一些包的支持，我们可以从 Oracle 的官方网站下载这些包。不同的应用程序需要不同的包的支持（请参考表 21-1）。

表 21-1　Oracle 即时客户端包的说明

包的名字	包的描述	支持的应用程序
Basic	运行 OCI、OCCI 和 JDBC-OCI 应用程序所需的所有文件	OCI OCCI JDBC
Basic Lite	Basic 的简化版，其中仅带有英文错误消息和 Unicode、ASCII 以及西欧字符集支持	OCI OCCI JDBC
JDBC Supplement	支持 XA、国际化和 JDBC 下的 RowSet 操作	JDBC
SQL*Plus	用于将 SQL*Plus 与即时客户端一起运行的额外的资料库和可执行文件	SQL*Plus
ODBC	用于支持 ODBC 应用程序的额外的库	ODBC
SDK	通过即时客户端开发 Oracle 应用程序的额外头文件和样例 Makefile	

续表

包的名字	包的描述	支持的应用程序
WRC	“负载重放客户端”用于 RAT 的 DB 重放特性的负载重放	
Precompiler	“proc”二进制文件的附属文件，以及用于预编译 Pro*C 应用程序和演示的相关文件	



21.2 使用即时客户端部署应用程序

应用程序开发完成后，我们就可以开始部署应用程序了。使用即时客户端部署应用程序的过程说明如下（本文以部署 Java 应用程序为例）。

STEP 01 到 Oracle 官方网站 http://www.oracle.com/technology/software/tech/oci/instantclient/index.html 下载所需要的包。

不同的应用程序需要不同的包，至于需要什么包，请参考表 21-1。

基于 Java 的应用程序通常需要 Basic、Basic Lite 两个包。Oracle 为不同的操作系统平台提供了包的不同版本，在下载时，需要注意你下载的包是否和你的操作系统兼容。

STEP 02 创建一个目录。

在要部署应用程序的机器上创建一个目录，该目录用于存放共享库（Shared Libraries）和 JDBC 类库（JDBC Class Libraries）。本例中创建目录 instantclient11，并假设其全路径是：

- /myapp/docinstantclient11(LINUX、Unix 平台)
- D:\myapp\docinstantclient11（Windows 平台）

STEP 03 把下载的文件解压到 STEP 02 创建的目录（instantclient11）中。

STEP 04 设置共享库（Shared Libraries）的环境变量，使它指向共享库的存放地。在 LINUX、Unix 平台，要设置环境变量 LD_LIBRARY_PATH；在 Windows 平台，要设置环境变量 PATH。

- export LD_LIBRARY_PATH=/myapp/docinstantclient11(LINUX、Unix 平台)
- PATH=D:\myapp\docinstantclient11（Windows 平台）

STEP 05 设置 JDBC 类库的路径 CLASSPATH，使它指向类库的存放地。本例中，CLASSPATH 的设置如下所示：

- export CLASSPATH=/myapp/docinstantclient11(LINUX、Unix 平台)
- CLASSPATH=D:\myapp\docinstantclient11（Windows 平台）

STEP 06 设置环境变量 TNS_ADMIN。

如果应用程序使用网络服务名连接数据库，则需要设置环境变量 TNS_ADMIN，TNS_ADMIN 指向文件 tnsnames.ora 所在的路径（本文假设也将该文件放在路径 instantclient11 下），TNS_ADMIN 的配置过程如下所示。

ⓐ 在设置环境变量之前，在目录 instantclient11 下创建文件 tnsnames.ora，文件中的

内容如下所示：

```
MYZDB =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = zero)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = zdb)
    )
  )
```

本例中的服务名是MYZDB。在这里指定要连接的数据库所在的主机名或者IP地址（本例中是 zero）、监听器的端口号（本例中是 1521）、实例名（本例中是 zdb）、所使用的协议（本例中是 TCP/IP）。

b 修改环境变量 TNS_ADMIN。

- export TNS_ADMIN=/myapp/docinstantclient11(LINUX、Unix 平台)
- TNS_ADMIN=D:\myapp\docinstantclient11（Windows 平台）

STEP 07 运行应用程序。

执行完前面的步骤后，如果应用程序的其他各方面都已准备就绪，就可以开始运行应用程序了。



本章总结

本章介绍了即时客户端的概念和如何使用即时客户端部署基于 Java 的应用程序，其他类型应用程序的部署过程也与此相同，只是它们需要不同的包而已。

第 22 章　数据库的备份

备份和恢复是 DBA 的重要职能，Oracle 提供的备份和恢复机制功能非常强大，它能使数据库发生灾难的情况下，也能把数据库恢复到灾难前的状态。

所谓备份，就是生成数据库中数据的副本，利用生成的副本可以重建数据。但是，这里的副本并不是简单的操作系统文件的复制。数据库备份分为物理备份（Physical Backup）和逻辑备份（Logical Backup）。

- 物理备份是对数据库的物理文件的备份，这些物理文件包括：数据文件、归档日志文件、控制文件等。可以把数据备份到磁盘上。
- 逻辑备份是利用 Oracle 的导出工具（Data Pump Export）导出数据库的逻辑对象（如表、存储过程等），把它们存放在二进制文件中，当发生数据丢失时，再利用 Oracle 的导入工具导入导出的数据。逻辑备份会导致数据的丢失，除非你只想用它备份历史数据，否则不要使用逻辑备份。

物理备份又被分类成冷备份（Cold Backup）和热备份（Hot Backup）。



22.1　数据库故障的种类

有很多情况都会导致数据库损坏和数据丢失，这些情况主要归成 3 大类：实例失败（Instance Failure）、介质失败（Media Failure）、用户错误（User Error）。



22.1.1　实例失败

硬件故障（如机器突然掉电）、操作系统软件故障（如操作系统突然宕掉）、Oracle 后台进程失败（Background Processes Fails），或者使用 SHUTDOWN ABORT 命令关闭数据库都会造成实例失败。

实例失败时，内存中的数据没有及时写到数据文件中，导致数据文件不一致。实例失败后需要进行实例恢复。

数据库启动时，Oracle 会自动执行实例恢复（Instance Recovery），实例恢复不需要用户的干预。执行实例恢复的原理是，应用重做日志文件使数据文件处于一致性状态。

22.1.2 介质失败

当 Oracle 不能读/写数据文件、控制文件、重做日志文件时，介质失败发生。介质失败可能由磁盘损坏或者操作系统故障所造成。发生介质失败后，DBA 需要还原丢失（损坏）的文件，然后执行恢复。

22.1.3 用户错误

用户错误是人为造成的，如用户不慎删除不应该删除的数据、用户误 DROP 掉一个表等。

22.2 冷备份（Cold Backup）

冷备份又叫脱机备份或者离线备份，指在数据库关闭的情况下，对数据库进行的备份。冷备份需要 DBA 备份所有数据文件、控制文件和重做日志文件。

22.2.1 一致性冷备份（Consistent Backup）

所谓一致性备份，指备份中的所有数据文件（Datafiles）和控制文件（Control Files）有相同的**系统改变号**（System Change Number，SCN）。冷备份必须是一致性备份（Consistent Backup），否则在数据库恢复后，有可能打不开数据库。

22.2.2 不一致冷备份（Inconsistent Backup）

冷备份要求 DBA 干净地关闭数据库（使用 SHUTDOWN NORMAL 或 SHUTDOWN IMMEDIATE 命令）。如果没有干净地关闭数据库（如使用 SHUTDOWN ABORT 命令）就对数据库进行备份，这样生成的备份是不一致备份（Inconsistent Backup）。不一致性冷备份的处理措施如下：

- 如果数据库运行在**归档模式**，且还存在冷备份后生成的归档日志文件，则可以把不一致冷备份恢复到一致性状态。
- 如果数据库运行在**非归档模式**，这样的备份是无效的备份，说白了，进行这样的备份还不如不备份。

22.3 热备份（Hot Backup）

热备份又叫联机备份，指数据库运行时对数据库进行的备份。对数据库进行热备份，

要求数据库运行在归档模式。热备份是不一致的备份，恢复数据时，需要应用归档日志文件，才能使数据库处于一致性状态，因此，需要数据库运行在归档模式。热备份需要备份数据文件、控制文件、归档日志文件等。

22.4 备份、恢复工具

有两种工具（方法）用来执行数据库的备份和恢复：一种方法是用户管理的备份和恢复（User-managed Backup And Recovery）；另外一种方法是使用 RMAN（中文叫“恢复管理器”）。

22.4.1 用户管理的备份和恢复

这是传统的备份和恢复的方法。通过这种方法，DBA 可以直接管理组成数据库的文件。这种方式通过 SQL*Plus 和操作系统命令的联合使用来完成数据库的备份和恢复。

“用户管理的备份和恢复”这个名字听起来怪怪的，这种方法以 SQL*Plus 为工具，因此，本书把这种方法叫“SQL*Plus 备份恢复方法”。

RMAN 功能强大，这是“SQL*Plus 备份恢复方法”无法比拟的地方。但是，为了利用 RMAN 的强大功能，RMAN 通常要求一个额外的数据库（Recovery Catalog），这是 RMAN 比较烦琐的地方。

22.4.2 RMAN

RMAN 是一个客户端工具，用于执行数据库的备份与恢复。RMAN 可以用来备份和恢复数据文件、归档日志文件和控制文件，也可以用来执行数据库的完全恢复或不完全恢复。总之，RMAN 功能强大！RMAN 的命令行提示符是 RMAN>。

1．通道（Channel）

通道表示到指定设备的一个数据流，一个通道对应一个服务器会话。如图 22-1 所示，通过在目标服务器上创建一个服务器会话，建立 RMAN 到目标数据库之间的一个通道，其中一个服务器会话把数据备份到磁盘存储，另外一个服务器会话把数据备份到磁带存储。

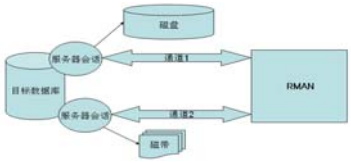


图 22-1 通道

用户可以手动分配通道，如 ALLOCATE CHANNEL c1 DEVICE TYPE sbt，也可以使

用 CONFIGURE CHANNEL 命令配置自动分配的通道。

2．恢复资料数据库（Recovery Catalog Database）

本节将介绍恢复资料数据库。有的书把“恢复资料”称为“恢复目录”。

🕮 恢复资料数据库简介

备份数据库时，会生成很多文件，如何记录这些文件的名字及位置等信息，成了一个难题。RMAN 中的**恢复资料数据库**正好帮助我们解决此问题。**恢复资料数据库**是一个**数据库**，用于存放**恢复资料（Recovery Catalog）**。

恢复资料，有的书叫“恢复目录”，就是一系列的表和视图，RMAN 利用这些表存放**备份信息**，执行备份和恢复时，RMAN 将访问这些表或视图。这些**备份信息**包括：

- 文件和归档日志文件的备份集（Backup Sets）和备份碎片的信息。
- 数据文件副本的信息。
- 归档日志文件及其副本的信息。
- 目标数据库的表空间和数据文件的信息。
- 存储脚本（Stored Scripts）。
- RMAN 的配置信息。

恢复资料被存放在**恢复资料数据库**的一个模式下，被一个用户所拥有。我们把这个用户叫**恢复资料拥有者（Recovery Catalog Schema Owner）**，这个用户其实就是一个普通的数据库用户，只是备份信息存放在它的账号之下。特别要注意的是，恢复资料不能驻留在要被备份的数据库（通常把这个数据库叫做**目标数据库**）中，否则如果目标数据库损坏，恢复资料也会损坏，也就无法执行数据库的恢复。

🕮 准备恢复资料

准备恢复资料通常要经历 3 个阶段：

- 第一阶段，创建**恢复资料的拥有者**。
- 第二阶段，创建**恢复资料**（其实就是创建相关的表和视图）。
- 第三阶段，注册**目标数据库**（其实就是把**目标数据库**的信息写到**恢复资料**中）。

🕮 第一阶段：创建恢复资料的拥有者

STEP 01 通过 SQL*Plus，以数据库管理员的身份登录到**恢复资料数据库**。

STEP 02 创建表空间。

创建一个表永久表空间，用于存放恢复资料

```
CREATE TABLESPACE space_for_backup
DATAFILE 'F:\newdb\zdb\space_for_backup1.dbf' SIZE 50M;
```

本例中创建表空间 space_for_backup，该表空间将用于存放恢复资料。

可选地，我们可以创建一个临时表空间，恢复资料的拥有者将使用该表空间。

STEP 03 创建恢复资料的拥有者。

```
CREATE USER rman IDENTIFIED BY zero
TEMPORARY TABLESPACE temp
DEFAULT TABLESPACE space_for_backup
QUOTA UNLIMITED ON space_for_backup;
```

恢复资料的拥有者是 rman（也可以使用其他名字），用户 rman 的默认表空间是 space_for_backup，临时表空间是 temp。

STEP 04 把恢复资料库的维护和查询权限授予恢复资料的拥有者 rman。

```
GRANT RECOVERY_CATALOG_OWNER TO rman;
```

第二阶段：创建恢复资料

创建**恢复资料**的实质就是创建一些表和视图。

STEP 01 在操作系统命令行中启动 RMAN。

```
C:\Documents and Settings\Administrator>rman
```

STEP 02 登录到恢复目录数据库（不是目标数据库）。

```
RMAN> CONNECT CATALOG rman/
恢复目录数据库口令:
连接到恢复目录数据库
RMAN>
```

以用户 rman 的身份登录**恢复目录数据库**，恢复目录数据库主要存放备份信息，恢复目录数据库和目标数据库是两个数据库（当然，也可以是相同的数据库，但是不推荐那么做，否则会有数据库不能恢复的风险）。

STEP 03 创建恢复资料。

```
RMAN> CREATE CATALOG TABLESPACE space_for_backup;
恢复目录已创建
```

创建恢复资料，恢复资料存放在表空间 space_for_backup 中。当然，也可以不指定表空间。如果不指定表空间，恢复资料将放在用户 rman 的默认表空间中。

STEP 04 在 SQL*Plus 中，使用下面的 SQL 语句查询表是否创建成功：

```
SELECT TABLE_NAME FROM USER_TABLES;
```

第三阶段：注册目标数据库

注册目标数据库的目的是把要备份的数据库（叫目标数据库）的信息（如 DBID 等）同步到**恢复资料数据库**中。

STEP 01 在操作系统命令行，启动 RMAN。

```
C:\Documents and Settings\Administrator>rman
```

STEP 02 登录到恢复资料数据库。

```
RMAN> CONNECT CATALOG rman/
恢复目录数据库口令:
连接到恢复目录数据库
RMAN>
```

STEP 03 登录到目标数据库（要被备份的数据库）。

```
RMAN> CONNECT TARGET SYS@myzdb;
目标数据库口令:
连接到目标数据库: ZDB (DBID=2235959226)
```

连接到目标数据库时，要使用专有服务器连接（即 SERVER = DEDICATED），否则注册时会产生错误“ORA-19550：无法在使用调度程序时使用备份/还原功能”。

STEP 04 注册目标数据库。

```
RMAN> REGISTER DATABASE;
```

STEP 05 查看注册是否成功。

```
RMAN> REPORT SCHEMA;
db_unique_name 为 ZDB 的数据库方案报表
永久数据文件列表
============================
文件大小 (MB) 表空间           回退段数据文件名称
---- -------- -------------------- ------- ------------------------
1    1024     SYSTEM               YES     F:\NEWDB\ZDB\SYSTEM01.DBF
2    1024     SYSAUX               NO      F:\NEWDB\ZDB\SYSAUX01.DBF
```

3．RMAN 的连接

■ 连接到目标数据库

```
RMAN> CONNECT TARGET sys@myzdb
目标数据库口令:
连接到目标数据库: ZDB (DBID=2235959226)
RMAN>
```

其中，TARGET 表示连接到目标数据库；sys 是目标数据中的数据库用户名；myzdb 指向目标数据库的网络服务名。

■ 连接到恢复资料数据库

```
RMAN> CONNECT CATALOG rman@bbc;
```

其中，CATALOG 表示连接到恢复资料数据库；RMAN 是恢复资料的拥有者；bbc 是网络服务名，指向恢复资料数据库。

22.5 冷备份的实施

冷备份是数据库关闭以后进行的备份。如果数据库运行在非归档模式，一定要执行数据库的一致性备份。一致性冷备份的过程说明如下。

STEP 01 执行下面的查询，得到数据库中所有数据文件的名字和位置。

```
SELECT NAME FROM V$DATAFILE;
```

STEP 02 得到控制文件的名字及位置。

```
SELECT NAME FROM V$CONTROLFILE;
```

STEP 03 干净地关闭数据库。

使用 SHUTDOWN NORMAL、SHUTDOWN IMMEDIATE 或者 SHUTDOWN TRANSACTIONAL 命令干净地关闭数据库。

STEP 04 备份文件。

使用操作系统命令，把数据库文件、控制文件、初始化参数文件等复制到备份的目标路径。

STEP 05 使用 STARTUP 命令重新启动数据库。

22.6 热备份的实施

热备份就是在线备份，它是在数据库运行时进行的备份，热备份生成的备份是不一致备份，只有执行恢复（Recover）才能使数据库处于一致性状态，这就会应用到归档日志文件，只有数据库运行在归档模式（ARCHIVELOG）下才会产生归档日志文件，因此，数据库必须运行在归档模式（ARCHIVELOG）下。

22.6.1 热备份的前提——归档管理

要执行热备份，数据库必须运行在归档模式。下面我们将介绍归档的概念。

1．归档的基本概念

归档是热备份的前提，只有数据库运行在归档模式（ARCHIVELOG）下，才可能找回丢失的数据。

当重做日志被填满以后，Oracle 将把填满的重做日志文件复制到一个或者多个路径下，这个过程叫归档，这样生成的文件叫归档日志文件（Archived Redo Log），存放归档日志文件的路径叫归档路径（或者叫归档目录）。只有数据库运行在归档模式下才能进行归档。归档的过程一般由归档进程（ARCH）自动完成，一个数据库可以有一个或者多个归档进程，归档进程的数量由初始化参数 LOG_ARCHIVE_MAX_PROCESSES 控制。归档重做日志文件的工作由归档进程完成，DBA 也可以手动归档重做日志文件。归档日志文件主要用于数据库的恢复。

数据库可以运行在两种模式：非归档模式（NOARCHIVELOG）和归档模式（ARCHIVELOG）。

2．使数据库运行在归档模式

通常，我们执行的备份都是热备份，热备份是不一致备份，要求数据库运行在归档模式。如何使数据库运行在归档模式，请参考下面的过程。

STEP 01 以用户 sys 的身份登录数据库。

```
SQL> CONNECT sys@myzdb as SYSDBA
```

STEP 02 查看数据库是否运行在归档模式。

```
SQL> ARCHIVE LOG LIST;
```

STEP 03 干净地关闭数据库。

```
SQL> SHUTDOWN IMMEDIATE;
```

STEP 04 修改初始化参数。

ⓐ 进入初始化参数文件所在的目录。

- $ORACLE_HOME/dbs/（UNIX）

➢ %ORACLE_HOME\database\(Windows)

其中，ORACLE_HOME 是环境变量，表示 Oracle 的主目录。PFILE 和 SPFILE 一般都存放在此路径下。本例中的路径是 C:\app\Administrator\product\11.1.0\db_3\database。

ⓑ 如果不存在初始化参数文件（PFILE），则可以根据 SPFILE 生成初始化参数文件（PFILE）。

```
    SQL>  CREATE  PFILE='C:\app\Administrator\product\11.1.0\db_3\database\
initzdb.ora'
    FROM SPFILE='C:\app\Administrator\product\11.1.0\db_3\database\spfilezdb.
ora'
```

其中，spfilezdb.ora 是 SPFILE，initzdb.ora 是 PFILE。

ⓒ 编辑初始化参数文件（PFILE）。

使用文本编辑器编辑文件 initzdb.ora，在该文件中，我们可以指定归档日志文件名的格式及归档路径等。

➢ 定义归档日志文件名的格式。

在归档日志文件名格式定义中，使用了变量%s 表示日志序列号（Log Sequence Number）；%t 表示线索号（Thread Number）；%d 表示数据库 ID（Database ID）；%r 表示重置日志 ID（Resetlogs ID）； %a 表示活动 ID 号（Activation ID）；%S 表示日志序列号，带有前导 0（即用 0 填充）；%T 表示线索号，带有前导 0（即用 0 填充）。

```
    log_archive_format='ARC%S_%R.%T'
```

➢ 设置归档路径。

设置第一个归档路径。

```
    LOG_ARCHIVE_DEST_1='LOCATION=D:\BACKUP\arch1'
```

设置第二个归档路径。

```
    LOG_ARCHIVE_DEST_2='LOCATION= D:\BACKUP\arch2'
```

在 Oracle 中，可以同时设置多个归档路径。LOG_ARCHIVE_DEST_1 表示第一个归档路径，LOG_ARCHIVE_DEST_2 表示第二个归档路径， LOG_ARCHIVE_DEST_*n* 表示第 *n* 个归档路径（*n* 最大为 10）。LOCATION 是关键字，指定归档路径的位置。也可使用 ALTER SYSTEM SET 命令对 LOG_ARCHIVE_DEST_*n* 的值进行动态修改。

ⓓ 重建（创建）SPFILE。

```
    SQL>  CREATE  SPFILE='C:\app\Administrator\product\11.1.0\db_3\database\
spfilezdb.ora'
    FROM   PFILE='C:\app\Administrator\product\11.1.0\db_3\database\initzdb.
ora'
    File created.
```

根据修改过的 PFILE（initzdb.ora）创建 SPFILE（spfilezdb.ora）。

STEP 05 启动数据库到 mount 状态。

```
SQL> STARTUP MOUNT
```



把数据库切换到归档模式。

```
SQL> ALTER DATABASE ARCHIVELOG;
```



查看数据库是否已经运行在归档模式。

```
SQL>ARCHIVE LOG LIST;
数据库日志模式            存档模式
自动存档                  启用
存档终点                  D:\BACKUP\arch1
最早的联机日志序列        158
下一个存档日志序列        161
当前日志序列              161
```



22.6.2 备份整个数据库

备份整个数据库的过程说明如下。



使用 RMAN 连接到目标数据库。

```
RMAN> CONNECT TARGET sys
```



连接到恢复资料数据库。

```
RMAN> CONNECT CATALOG rman;
```



备份整个数据库。

```
BACKUP DATABASE FORMAT 'D:\BACKUP\bk0_%s_%p_%t';
```

FORMAT 指定备份生成的文件的路径及文件名的格式；%s 指定备份集；%p 指定碎片编号（Piece Number）；%t 指定时间戳。

22.6.3 备份单个表空间

本文以备份表空间 secondspace 为例讲述如何备份单个表空间，有两种方法可用于备份单个表空间，详细过程如下所示。

■ 方法 1：使用 SQL*Plus 备份单个表空间

STEP 01 查询要备份的表空间由哪些数据文件组成。

```
SQL> SELECT TABLESPACE_NAME, FILE_NAME
  2  FROM SYS.DBA_DATA_FILES
  3  WHERE TABLESPACE_NAME = 'SECONDSPACE';
TABLESPACE_NAME     FILE_NAME
-----------------------------------------
SECONDSPACE         F:\NEWDB\ZDB\SECONDSPACE1
```

本例中，表空间 secondspace 由数据文件 secondspace1 组成。

STEP 02 把表空间置于备份模式。

```
SQL> ALTER TABLESPACE secondspace BEGIN BACKUP;
表空间已更改。
```

STEP 03 使用操作系统命令把组成表空间的数据文件复制到备份路径中。

STEP 04 让表空间退出备份模式，回到正常状态。

```
SQL> ALTER TABLESPACE secondspace END BACKUP;
表空间已更改。
```

STEP 05 执行手工归档。

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

■ 方法 2：使用 RMAN 备份单个表空间

```
BACKUP TABLESPACE secondspace FORMAT 'D:\BACKUP\bk0_%s_%p_%t';
```

22.6.4 备份单个数据文件

不能使用 SQL*Plus 直接备份单个数据文件，但是可以通过 SQL*Plus 间接备份单个数据文件。

下面将使用 RMAN 备份数据文件，使用 RMAN 备份数据文件的例子如下：

```
BACKUP DATAFILE 'F:\NEWDB\ZDB\SECONDSPACE1' FORMAT 'D:\BACKUP\bk0_%s_%p_%t';
```

本例使用 RMAN 备份数据文件 f:\newdb\zdb\secondspace1。

22.6.5 备份归档日志文件

除了备份数据文件之外，归档日志文件也需要进行备份。有两种方法可用于归档日志

文件的备份。

■ 方法 1：使用操作系统命令直接备份



查看数据库中有哪些归档日志文件。

```
SELECT THREAD#,SEQUENCE#,NAME
FROM V$ARCHIVED_LOG;
```



使用操作系统命令把归档日志文件从归档路径复制到备份目录。

■ 方法 2：使用 RMAN 进行备份

```
BACKUP ARCHIVELOG ALL FORMAT 'D:\BACKUP\bkar_%U';
```

其中，ALL 表示备份所有的归档日志文件；FORMAT 指定备份生成文件的格式；%U 表示将产生唯一文件名（也就是文件名不会重复）。



22.6.6 备份控制文件

可以使用下面几种方法备份控制文件。

■ 方法 1：使用 SQL*Plus 备份控制文件

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO 'D:\BACKUP\conf.bak';
数据库已更改。
```

■ 方法 2：把控制文件备份到跟踪文件中，在跟踪文件中将生成创建控制文件的脚本

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
数据库已更改。
```

■ 方法 3：使用 RMAN 备份控制文件

```
RMAN> BACKUP CURRENT CONTROLFILE;
```

■ 方法 4：启用控制文件的自动备份，当使用 RMAN 执行备份的时候，RMAN 会自动备份控制文件（也会自动备份 SPFILE）

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
```



22.6.7 备份 SPFILE

使用 RMAN 备份 SPFILE 的命令如下。当然，我们也可以直接使用操作系统命令把 SPFILE 复制到备份路径，这也算是 SPFILE 的一种备份方式。当数据库的 SPFILE 丢失时，我们可以把备份的 SPFILE 复制回来。

```
RMAN> BACKUP SPFILE;
```



本章总结

本章除了介绍 Oracle 备份的概念之外，还介绍了备份工具 RMAN。在制定数据库的备份恢复策略时，不允许你出现任何差错，一失足成千古恨的后果我们是承担不起的！因此，你需要深刻理解本章提到的概念。在实施备份策略时，你至少要测试一次你的备份文件是否可以用于恢复数据库。

第 23 章　数据库的恢复

第 22 章我们讲解了数据库的备份，本章我们将讲解数据库的恢复。数据库故障的不同或者备份策略的不同，将影响数据库的恢复手段。本章将讲解数据冲突的恢复、冷备份的恢复和热备份的恢复。



23.1　数据冲突的修复

数据恢复向导（Data Recovery Advisor，DRA）是帮助诊断和修复数据故障的工具。它根据症状对故障进行分析和诊断，并智能地确定最佳修复策略。该工具还可自动诊断和修复故障。

DRA 可以诊断下面的错误：

- 数据库文件或者控制文件因为不存在、离线或者权限等问题而不能被访问。
- 块因为各种原因引起的冲突。
- 数据文件与其他文件的不一致性。
- 由硬件错误、操作系统错误等引起的 I/O 失败。

数据故障检测和修复的基本步骤说明如下。


对数据库进行验证。

```
RMAN> VALIDATE DATABASE;
```

RMAN 对数据库进行验证，如果发现错误，会把这些错误记录到自动诊断库（Automated Diagnostic Repository，ADR）中。


使用 LIST FAILURE 命令列出故障。

```
RMAN> LIST FAILURE;
Failure ID Priority Status Time Detected Summary
---------- -------- --------- ------------- -------
142 HIGH OPEN 23-APR-09 One or more non-system datafiles
are missing
101 HIGH OPEN 23-APR-09 Datafile 1:
'D:\mydata\psdd.dbf' contains one or more corrupt blocks
```

本例中，有两个故障，故障编号是 142、101。

STEP 03 根据故障编号，获得故障的详细信息。

```
RMAN>LIST FAILURE 101 DETAIL;
```

101 是 STEP 02 获得的故障编号。

STEP 04 得到故障的修复建议。

```
RMAN> ADVISE FAILURE;
```

此命令将给出故障的修复建议。

也可列出指定故障的修复建议。

```
RMAN> ADVISE FAILURE 101;
```

本例列出故障 101 的修复建议。

STEP 05 执行故障修复。

```
RMAN> REPAIR FAILURE;
```

STEP 06 再次查看故障是否已经被修复。

```
RMAN> LIST FAILURE;
```

23.2 冷备份的恢复

所谓“冷备份的恢复”是指利用冷备份恢复整个数据库，在非归档模式（NOARCHIVELOG）下产生的冷备份必须是一致性备份，否则数据库恢复将面临失败。下面，我们将使用 SQL*Plus 执行冷备份的恢复，详细的过程说明如下。

STEP 01 如果数据库还在运行，需要关闭数据库。

```
SHUTDOWN IMMEDIATE
```

STEP 02 还原数据库的备份。

使用操作系统命令把数据库的备份复制到原来的位置。

STEP 03 把数据库启动到装载状态。

```
STARTUP MOUNT
```


STEP 04 重命名数据文件。

如果备份的数据文件被复制到与原来不同的位置，需要修改控制文件，使控制文件知道这种位置的变化，否则不需要做任何改变。

```
ALTER DATABASE RENAME FILE 'D:\mydata\psdd.dbf' TO
'F:\mydata\psdd2.dbf';
```

D:\mydata\psdd.dbf 是原来数据文件的名字及位置，F:\mydata\psdd2.dbf 是数据文件新的名字及位置。

STEP 05 重命名重做日志文件。

如果重做日志文件原来所在的磁盘还没有修好，或者想把它们放在其他位置，可以通过重命名重做日志文件来实现。

```
ALTER DATABASE RENAME FILE 'D:\mydata\redo01.log' TO
'F:\mydata\redo_01_1.log';
```

D:\mydata\redo01.log 是原来重做日志文件的名字及位置，F:\mydata\redo_01_1.log 是重做日志文件新的名字及位置。

STEP 06 执行数据库的不完全恢复。

```
RECOVER DATABASE UNTIL CANCEL
CANCEL
```

这样做是为了使 Oracle 能够重新设置重做日志文件。

STEP 07 用选项 RESETLOGS 打开数据库。

```
ALTER DATABASE OPEN RESETLOGS;
```

RESETLOGS 将使 Oracle 重新设置重做日志文件。

23.3 热备份的恢复

在数据库打开的情况下，产生的备份叫热备份，热备份是数据库备份的主要方式。接下来，我们将讨论如何利用热备份恢复数据库。

用热备份恢复数据库通常都会经历两大过程：

- 还原数据库（Restore Database）
- 恢复数据库（Recover Database）

我们经常会混淆“还原数据库”和“恢复数据库”的概念，这其实是两个过程，“还原数据库”是指把数据库的备份复制到原来的位置，“恢复数据库”是指在还原数据库后，应用归档日志文件（备份以后产生的）或者重做日志文件的内容。

23.3.1 恢复整个数据库

如果整个数据已经瘫痪，我们可以恢复整个数据库。使用 RMAN 恢复整个数据库的过程如下所示。

STEP 01 如果数据库在运行，则强制把数据库启动到装载状态。

```
RMAN> STARTUP FORCE MOUNT;
```

STEP 02 还原数据库。

```
RMAN> RESTORE DATABASE;
```

STEP 03 恢复数据库。

```
RMAN> RECOVER DATABASE;
```

STEP 04 打开数据库。

```
RMAN> ALTER DATABASE OPEN;
```

23.3.2 恢复单个表空间

如果只是某个表空间坏掉，我们不需要关闭数据库，在数据库运行的情况下，我们也能恢复丢失的数据，详细过程说明如下。

STEP 01 使要被恢复的表空间离线。

```
RMAN> SQL 'ALTER TABLESPACE users OFFLINE';
```

STEP 02 运行 RMAN 的 RUN 命令，执行表空间的还原与恢复。

```
RUN
{
SET NEWNAME FOR DATAFILE 'D:\mydata\users01.dbf'
TO 'F:\mydata\users01.dbf'; #重命名数据文件
RESTORE TABLESPACE users; #还原表空间
SWITCH DATAFILE ALL; #更新控制文件
RECOVER TABLESPACE users; #执行表空间的恢复
}
```

知识点索引——RMAN 命令 RUN

RUN 是 RMAN 的命令，它把一系列的 RMAN 命令集成到一个块中，然后顺序执行这些命令。这样的方式有利于备份和恢复代码的模块化设计。RUN 代码块以“{”开头，以“}”结尾。{}中的语句可以作为一个代码块一起执行。

STEP 03 使表空间重新在线。

```
RMAN> SQL 'ALTER TABLESPACE users ONLINE';
```

23.3.3 恢复单个数据文件

如果单个数据文件损坏，不必恢复整个表空间，更不必恢复整个数据库，只需恢复损坏的数据文件即可。单个数据文件恢复的步骤说明如下。

STEP 01 从备份中还原数据文件 D:\mydata\users01.dbf。

```
RMAN> RESTORE datafile 'D:\mydata\users01.dbf';
```

STEP 02 执行数据文件 D:\mydata\users01.dbf 的恢复。

```
RMAN> RECOVER DATAFILE 'D:\mydata\users01.dbf';
```

STEP 03 使数据文件 D:\mydata\users01.dbf 在线。

```
SQL> ALTER DATABASE DATAFILE 'D:\mydata\users01.dbf' ONLINE;
```



23.3.4 修复数据块

如果数据块冲突，不必关闭数据库，可以在线修复冲突块。修复冲突块的过程如下所示。



找出冲突块。

有两种方法可以知道数据块中的冲突块。

方法 1：查询视图 V$DATABASE_BLOCK_CORRUPTION

```
SQL> SELECT * FROM V$DATABASE_BLOCK_CORRUPTION;
```

方法 2：从跟踪文件或者警报文件中进行查找

冲突块的信息会记录在跟踪文件（Trace Files）或者警报文件（Alert Log）中，从这些文件中，我们可以了解冲突块的信息。至于跟踪文件和警报文件的位置，我们可以通过下面的查询获得。

```
SQL> SELECT NAME,VALUE FROM V$DIAG_INFO;
```



修复冲突块。

```
RMAN> BLOCKRECOVER
2> DATAFILE 7 BLOCK 3
3> DATAFILE 2 BLOCK 235;
```

修复数据文件 7 中的坏块 3 及数据文件 2 中的坏块 235。

也可以使用下面的命令修复冲突块：

```
RECOVER CORRUPTION LIST;
```

不过，该命令会修复视图 V$DATABASE_BLOCK_CORRUPTION 中列出的所有冲突块。



本章总结

本章介绍 Oracle 数据库的恢复。数据库故障的不同，恢复方式也不同。需要注意的是，恢复的前提是备份，如果没有完善的备份机制，即使“巧妇”也难为无米之炊。

第 24 章　项目实战之——网银子系统

不深入实际就没有发言权，前面的章节，我们准备了好多的理论知识，只期盼有一天能够派上用场！在这里，我们将给你一个实战的机会！本章的例子来自一个网上银行的项目，我们对该项目进行了简化，只是给你展示该系统的部分功能！但是我们希望对于刚进入软件开发的朋友来说，也许这是一个好的开端！在讲述“网银子系统”之前，请容许我们讨论一下软件生命周期。



24.1　软件生命周期（System Development Life Cycle）

在软件开发中，我们把软件生命周期分成 6 个阶段（当然，还有其他划分方法）：

1．可行性分析

通过调研，与需求方进行沟通，从资金、技术、时间、人力资源等各方面论证项目的可行性。可行性分析是 SDLC 的第一步，如果一个项目还没有经过严格的论证就匆匆进入开发，可能我们收获的只有失败和各种资源的浪费！

2．需求分析

在确定软件开发可行的情况下，应对软件需要实现的各个功能进行详细分析。在需求分析阶段，也许客户还不是很了解自己想要的东西，这时候，我们要学会深度挖掘客户需求。客户需求越明了，软件开发的风险越小！需求分析不彻底是项目失败的重要原因之一（付出高昂代价也算一种失败）。

3．架构设计

客户需求确定以后，要对整个系统进行架构设计，包括软件系统框架设计、数据库模型设计、硬件框架结构设计。设计的思想是由宏观到微观，由粗到细。无论是哪种设计，都应该把影响性能的因素考虑进去。

4．编码

编码的目的是将软件系统框架设计的结果转换成计算机可运行的程序代码。在编码阶段，要有统一的命名规范，要注意程序的可读性和可维护性。更重要的是，把性能放在编码所要考虑的首位。

5．测试

通常，软件开发和软件测试同时进行。软件测试的目的是使软件符合“需求分析”和“架构设计”的要求。需要注意的是，软件测试的目的是发现问题，而不是规避问题。软件测试有白盒测试和黑盒测试。

6．运行和维护

一旦软件投入运营，将会发现一些问题，这是软件生命周期不可逃避的一个阶段。软件的维护包括纠错性维护和改进性维护两个方面。在这个阶段，如果遇到“可行性分析”、“需求分析”、“架构设计”阶段忽略的问题，对于一个项目来说，这可能是一个致命的打击，因此，这三个阶段的工作极其重要！



注意：对于“网银子系统”的软件生命周期中，由于篇幅所限，我们并不会对每个阶段进行详细的讲解，甚至会忽略软件生命周期的某些阶段。我们只是想让读者了解软件开发的大致过程而已。



24.2 业务需求分析——前期工作

本节主要讲述需求分析。要注意的是，本书省略了“可行性分析”阶段。



24.2.1 业务需求

网银子系统是银行业务系统的一个分支。它主要负责提供客户在银行柜台或者 ATM 机中的存取款操作。系统自动记录用户的每笔交易操作，并提供给用户查询交易明细的功能，使用户能够详细浏览某个范围内的交易明细。



24.2.2 系统效果展示

在 IE 地址栏中输入 http://localhost:8080/banktrade，进入系统登录界面，如图 24-1 所示，输入证件号码（内置的是 001 和 002）、登录密码（都是 banktrade）、附加码，单击“登录”按钮进入用户主界面，如图 24-2 所示。

图 24-2 显示了用户的账户、账户余额、身份证号、开户时间、用户姓名等信息，单击“RMB 业务”按钮进入 RMB 交易操作界面，如图 24-3 所示。

用户在“金额（Y）”中填入需要交易的金额，然后选择交易类型：支出或存入，最后选择交易方式，单击“确定”按钮，交易完成后，系统会自动刷新用户的余额。

在如图 24-2 所示的界面上单击“账户明细”按钮，进入交易日志界面，用户选择“起止日期”范围后，单击“确定”按钮，界面将列出此日期范围内的交易明细，如图 24-4 所示。



图 24-1　用户登录界面

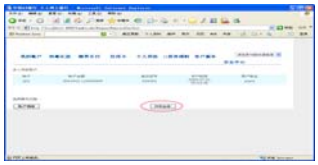


图 24-2　用户账户信息



图 24-3　人民币交易

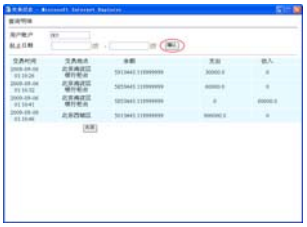


图 24-4　明细显示

24.3　架构设计——架起业务与软件的桥梁

架构设计包括软件系统框架设计、数据库模型设计、硬件框架结构设计。本章只讲诉软件系统框架设计和数据库模型设计。

24.3.1　软件系统框架设计

本节将主要讲述系统模块划分和各个模块的功能。

“网银子系统”包括如下几大模块：

- 用户开户：由于篇幅原因，系统不实现用户开户功能，但事先预定义了两个用户：① 用户昵称/证件号码：001，密码：banktrade；② 用户昵称/证件号码：002，密码：banktrade。
- 账户信息展示：在页面上展示用户账号的相关信息。
- 用户存取款：用户通过银行柜台或者 ATM 进行存取款业务操作。
- 交易明细：用户查看某段时间内自己账户的交易明细。

“网银子系统”的流程如图 24-5 所示。

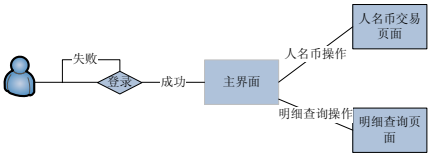


图 24-5　系统流程

24.3.2　数据库模型设计

本节将按照概念模型→逻辑模型→物理模型的顺序描述“网银子系统”的数据库模型设计。

1．概念模型（Conceptual Data Model）

概念模型是从非技术的角度描述客观世界的事或物。这种模型完全不涉及信息在计算机系统中的表示，更多是从业务角度来考虑。“网银子系统”中存在客户，客户要进行交易，交易时涉及交易地点。

2．逻辑模型（Logical Data Model）

逻辑模型是从**数据库的角度**去描述现实世界中存在的事或物。逻辑模型是在概念模型的基础上生成的。逻辑模型最常用的表示方法是“实体-关系图（Entity Relationship Diagram，E-R 图）”。构成 E-R 图的基本要素是实体（Entity）、属性（Attribute）和关系（Relationship）。

实体

“网银子系统”涉及 3 个实体：用户、交易、交易地址。它们三者的 E-R 图如图 24-6 所示。

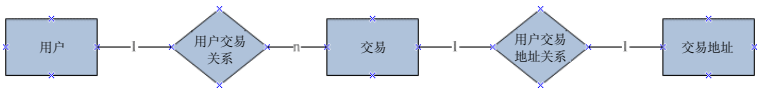


图 24-6　E-R 图

关系

图 25-6 展示了系统中各个实体间的关系：

- “用户”和“交易”之间是一对多的关系，因为每个用户会进行多次 RMB 交易操作。
- “交易”和“交易地址”之间是一对一的关系，用户的每次 RMB 交易只能存在一个地点。

📖 属性

“用户”实体、“交易”实体、“交易地址”实体的属性如图 24-7 所示。

- “用户”实体的属性是：卡号、密码、用户姓名、身份证、开户时间、余额、联系方式、地址。
- “交易”实体的属性是：业务 ID、卡号、交易地址 ID、交易数量、交易日期、交易方式、用户余额。
- “交易地址”实体的属性是：交易地址 ID、交易地址。

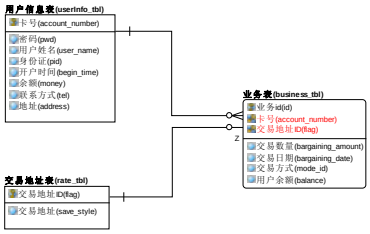


图 24-7　用户属性和表

如图 24-7 所示，实体和表的对应关系如下所示：

- “用户”实体对应用户信息表（userinfo_tbl）。
- “交易”实体对应业务表（business_tbl）。
- “交易地址”实体对应交易地址表（rate_tbl）。

3．物理模型（Physical Data Model）

从逻辑模型可以生成物理模型。逻辑模型与任何数据库平台没有关系，而物理模型对应特定的数据库平台，如开发基于 Oracle 的应用，物理模型就与 Oracle 数据库相关。通常，只要设计人员设计好逻辑模型，开发人员就可以使用数据模型设计工具（如 PowerDesigner、ERWin）从逻辑模型自动生成物理模型。生成物理模型后，我们可以使用诸如 PowerDesigner 等工具直接生成建表的 SQL 语句（请参考本书配套光盘中的 banktrade\banksystem.sql）。

4．模型设计工具

在设计数据模型时，会经历概念模型设计→逻辑模型设计→物理模型设计的过程。当前主流的模型设计工具有 PowerDesigner、ERWin 等。PowerDesigner、ERWin 把概念模型设计和逻辑模型设计合并成一个阶段。有关 PowerDesigner、ERWin 的使用，请参考相关书籍或者它们的官方文档。

24.4 编码

本文省略了编码的过程。有关的代码放在本书配套光盘中。在本章的后面会有主要代码的解释。如果你想进一步了解开发方面的知识，请参考 Java 开发的相关书籍。

24.5 测试

在编码的过程中要进行测试，有关测试的技术，请参考相关书籍，本书将省略测试的过程。

24.6 把理想变成现实

在软件开发完成后，就可以开始实施工作了。实施工作涉及 JDK 的安装、Tomcat 的安装、网银子系统的配置及服务的启动。

24.6.1 安装 JDK

本节主要讲述 JDK 的安装和配置。先要安装 JDK，然后设置和 JDK 相关的环境变量。

1．JDK 安装图解

工欲善其事，必先利其器，想让 Java 程序运行起来，必须先安装 Java 运行环境，安装步骤说明如下。

STEP 01 下载 JDK 安装程序。

到 Sun 公司网站下载 JDK，下载地址是：http://JAVA.sun.com/JAVAse，本例下载 JDK1.6，下载的文件名是 jdk-6u17-windows-i586.exe。

STEP 02 启动 JDK 安装。

双击安装文件 jdk-6u17-windows-i586.exe，出现如图 24-8 所示的界面。

STEP 03 接受许可证协议。

单击 接受(A) > 按钮，接受 JDK 安装许可证协议，出现如图 24-9 所示的界面。

STEP 04 JDK 安装路径选择。

在此，我们可以选择要安装哪些程序功能，要忽略哪些程序功能。除此之外，我们还可以单击 更改(A)... 按钮，修改安装路径。单击 下一步(N) > 按钮，程序自动安装（如图 24-10 所示），安装完成后，出现如图 24-11 所示的界面。



图 24-8　许可证协议



图 24-9　JDK 安装路径选择

STEP 05　安装 JRE 环境。



图 24-10　正在安装



图 24-11　JRE 安装

在此，要安装 Java 运行环境（JRE），如果想更改安装路径，请单击 更改(A)... 按钮。单击 下一步(N) > 按钮，出现如图 24-12 所示的界面，显示 JDK 的安装进度。



注：JDK 是 Java 编译环境，JRE 是 Java 运行环境。

STEP 06　安装进度显示。

STEP 07　程序安装完成后，出现如图 24-13 所示的界面。



图 24-12　JDK 安装进度



图 24-13　JDK 安装完成

STEP 08 单击 完成(F) 按钮，JDK 安装完成。

2．设置环境变量

想要在 Windows 环境中开发和运行 Java 项目，还必须设置 Windows 环境变量，这样才能通过命令来编译和运行 Java 程序，下面通过图解的方式，详细讲解环境变量的设置。

STEP 01 首先右键单击“我的电脑”→“属性”，出现如图 24-14 所示的窗口。

STEP 02 选择“高级”菜单选项，出现如图 24-15 所示的界面。



图 24-14　系统属性

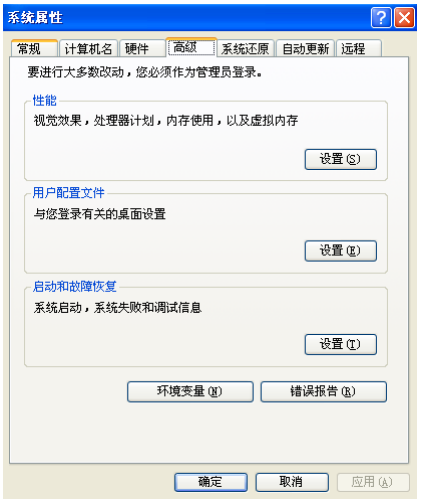


图 24-15　高级界面

STEP 03 新建环境变量 JAVA_HOME。

ⓐ 单击“环境变量”按钮，在“系统变量”下单击“新建”按钮，新建“JAVA_HOME”变量，变量值为 JDK 的安装目录，本例中是 C:\Program Files\Java\jdk1.6.0_13，如图 24-16 所示。

ⓑ 单击“确定”按钮，添加完成。

STEP 04 新建环境变量 CLASSPATH。

ⓐ 单击“新建”按钮，新建变量名为“CLASSPATH”变量，变量值为 JDK 安装目录下的 dt.jar 和 tools.jar 两个包的路径，默认为 %JAVA_HOME%\dt.jar;%JAVA_HOME%\tools.jar，这两个路径值之间用分号隔开，如图 24-17 所示。



图 24-16　新建 JAVA_HOME

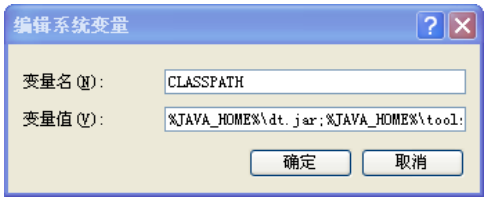


图 24-17　新建 CLASSPATH

ⓑ 单击“确定”按钮，添加完成。

STEP 05 修改环境变量 PATH 的值

ⓐ 选择环境变量“PATH”，单击“编辑”按钮，修改 PATH 变量的值，添加 JDK 的 bin 目录，默认为%JAVA_HOME%\bin，如图 24-18 所示。



图 24-18　编辑 PATH 变量

ⓑ 单击“确定”按钮，添加完成。

24.6.2　安装 Tomcat

本节主要讲述 Tomcat 的安装和配置。先要安装 Tomcat，然后对其环境进行配置。

1．Tomcat 安装图解

Tomcat 是一款免费、开源的 Web 服务器，用来发布 B/S 项目。Tomcat 的安装过程如下所示。

STEP 01 下载 Tomcat 安装程序。

从网站 http://tomcat.apache.org/下载 tomcat 的安装程序。

STEP 02 启动 Tomcat 安装。

鼠标双击安装程序 apache-tomcat-6.0.20.exe，出现如图 24-19 所示的界面。

单击 Next > 按钮，出现如图 24-20 所示的界面。

STEP 03 许可证协议。

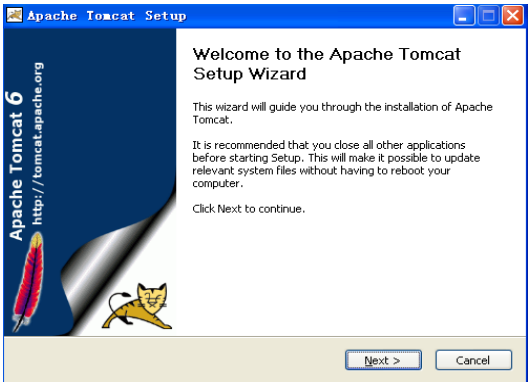


图 24-19　Tomcat 启动安装

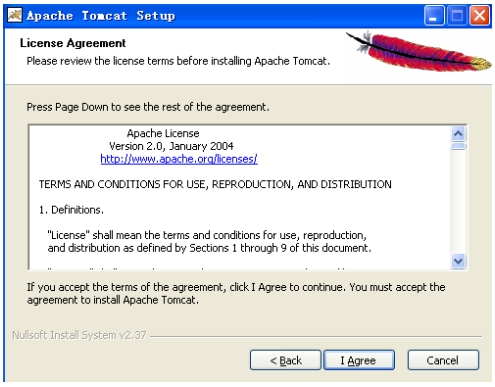


图 24-20　Tomcat 安装

同意 Tomcat 安装许可协议，单击 I Agree 按钮，出现如图 24-21 所示的界面。

STEP 04 安装选项。

在此，选择要安装的选项，如选择安装 Tomcat 文档、样例。通常选择默认设置，单击 Next > 按钮，出现如图 24-22 所示的界面。

STEP 05 选择安装路径。

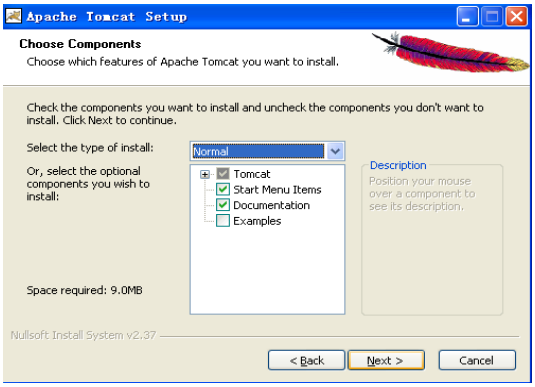


图 24-21 Tomcat 安装选项

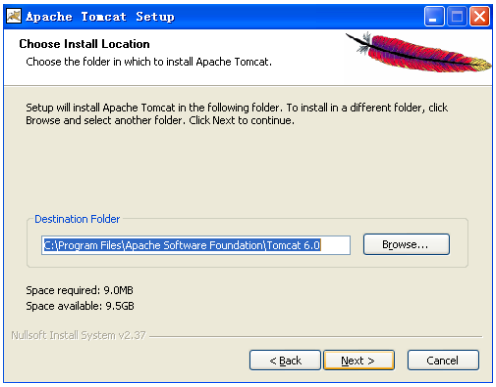


图 24-22 选择安装路径

选择 Tomcat 的安装路径。本书保持默认设置，单击 Next > 按钮，出现如图 24-23 所示的界面。

STEP 06 端口、管理员设置。

在此，设置 Tomcat 启动的端口号以及进入管理界面的用户名和密码，通常都选择默认值，单击 Next > 按钮，出现如图 24-24 所示的界面。

STEP 07 选择 Java 虚拟机的路径。

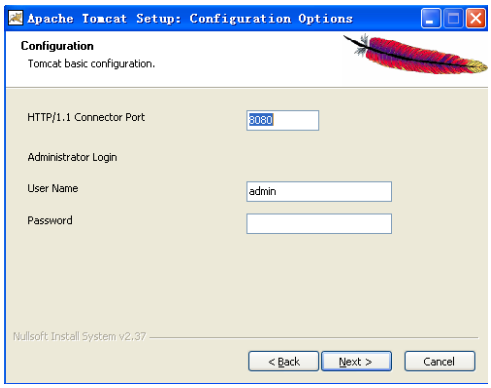


图 24-23 Tomcat 端口设置

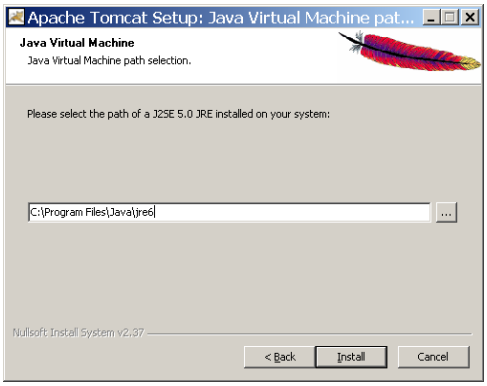


图 24-24 设置 Java 虚拟机的路径

在此，选择 Java 虚拟机的路径（此路径就是 JRE 安装时的路径），然后单击 Install 按钮，安装开始。

STEP 08 安装进度如图 24-25 所示。

STEP 09 安装结束后出现如图 24-26 所示的界面。

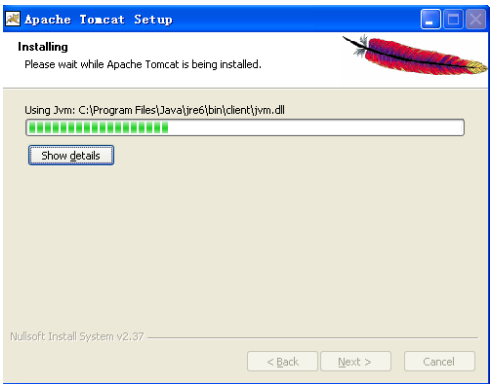


图 24-25　Tomcat 安装进度



图 24-26　Tomcat 安装完成

STEP 10　程序安装完成

在如图 24-26 所示的界面中去掉第一个选项，单击 Finish 按钮，程序安装完成。

2．设置环境变量

Tomcat 环境变量的配置过程如下所示。

STEP 01　新建环境变量 CATALINA_HOME

ⓐ 右键单击“我的电脑”→“属性”→“高级”→“环境变量”，出现系统环境变量列表框，单击系统变量下面的“新建”按钮，新建 “CATALINA_HOME”变量，变量值为 Tomcat 的安装路径，默认是 C:\Program Files\Apache Software Foundation\tomcat6.0，如图 24-27 所示。

ⓑ 单击“确定”按钮，添加完成。

STEP 02　新建环境变量 TOMCAT_HOME。

ⓐ 单击“新建”按钮，新建“TOMCAT_HOME”环境变量，变量值为 Tomcat 的安装路径，默认是 C:\Program Files\Apache Software Foundation\tomcat6.0，如图 24-28 所示。



图 24-27　设置 CATALINA_HOME 变量



图 24-28　设置 TOMCAT_HOME 变量

ⓑ 单击“确定”按钮，添加完成。

STEP 03　修改环境变量 CLASSPATH。

ⓐ 选择列表中的“CLASSPATH”变量，单击“编辑”按钮，添加 Tomcat 安装目录下 lib 中 servlet-api.jar 和 jsp-api.jar 的路径，CLASSPATH 的默认值是%TOMCAT_HOME%\common\lib\servlet-api.jar;TOMCAT_HOME%\common\lib\jsp-api.jar（中间用分号隔开），如图 24-29 所示。

ⓑ 单击“确定”按钮，修改完成。

STEP 04 修改环境变量 Path 的。

ⓐ 选择“Path”变量，单击“编辑”按钮，添加 Tomcat 安装目录下 bin 路径，默认是%TOMCAT_HOME%\bin，如图 24-28 所示。

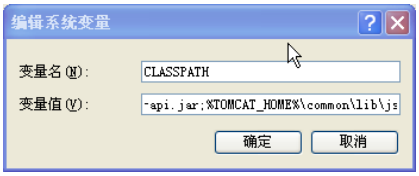


图 24-29　设置 CLASSPATH 变量

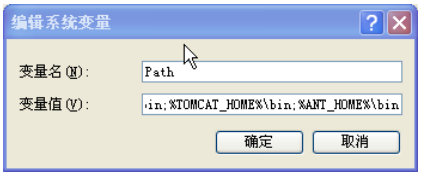


图 24-30　设置 PATH 变量

ⓑ 单击“确定”按钮，修改完成。

3．测试 Tomcat 的安装是否正确

STEP 01 首先进入到 Tomcat 安装目录的 bin 目录中，鼠标双击“startup.bat”文件，启动 Tomcat 服务，如图 24-31 所示。

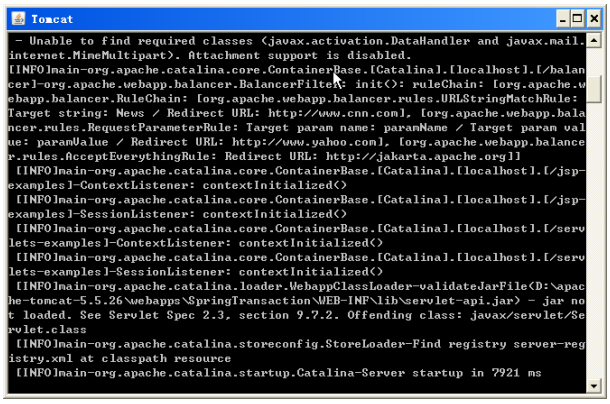


图 24-31　启动 Tomcat

STEP 02 打开 IE 浏览器，输入 http://localhost:8080，然后按回车键，如果出现 Tomcat 的欢迎界面（如图 24-32 所示），恭喜你，Tomcat 安装成功！



图 24-32　Tomcat 主界面

24.6.3 网银子系统配置

JDK 和 Tomcat 是网银子系统运行的基础。安装完成 JDK、Tomcat 之后，下面才真正进入网银子系统的配置。其配置过程包括以下几个步骤：

- 准备数据库表和样例数据
- 把代码从光盘复制到主机
- 配置数据库连接
- 系统启动脚本配置
- 启动服务
- 验证系统是否成功运行

1．准备数据库表和样例数据

本例假如你已经创建数据库 ZDB，数据库服务器的 IP 地址是 192.168.1.114。并且你已经配置好监听器，监听的端口是 1521（默认端口）。

STEP 01 打开 SQL*Plus，以管理员身份登录数据库。

```
C:\Documents and Settings\Administrator>sqlplus
SQL*Plus: Release 11.1.0.6.0 - Production on 星期二 12月 8 17:24:53 2009
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
请输入用户名:  system
输入口令:
连接到:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
```

STEP 02 创建用户 user4，网银子系统的数据放在用户 user4 之下。

```
SQL> CREATE USER user4 IDENTIFIED BY pwd4;

用户已创建。
```

STEP 03 授予用户 user4 相应的权限。

```
SQL> GRANT CREATE SESSION TO user4;
授权成功。

SQL> GRANT CREATE TABLE TO user4;
授权成功。

SQL> GRANT UNLIMITED TABLESPACE TO user4;
授权成功。
```

STEP 04 以 user4 的身份登录数据库。

```
SQL> connect user4
```

```
输入口令:
已连接。
```



运行创建表的脚本 banksystem.sql，并导入数据。

```
SQL>@F:\banktrade\banksystem.sql
```

本例假设 F 盘就是光盘，创建表的脚本 banksystem.sql 放在光盘的目录 banktrade 中。

2．把代码从光盘复制到主机

在 D 盘创建目录 banktrade_code，将本书配套光盘中 banktrade\banktrade_code 下的内容全部复制到 D:\ banktrade_code\。

3．配置数据库连接

系统引用 Spring 框架自带的数据源对象（`BasicDataSource`），数据源对象用于连接数据。配置时，开发人员只需修改配置文件 D:\banktrade_cod\WebRoot\WEB-INF\classes\ApplicationContext.xml，以下是该文件的样例（节选）：

```
<!-引用 Spring 框架中的 BasicDataSource 对象-->
<bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName"
            value="oracle.jdbc.driver.OracleDriver"/>
<!-数据库 IP、端口、数据库名称 -->
        <property name="url"
            value="jdbc:oracle:thin:@192.168.1.114:1521:zdb" />
    <!-登录数据库时的用户名和密码 -->
        <property name="username" value="user4" />
        <property name="password" value="pwd4" />
    </bean>
```

4．系统启动配置文件

在 Tomcat 的 localhost 路径下（本例中的全路径是 C:\Program Files\Apache Software Foundation\tomcat6.0\conf\Catalina\localhost）创建文件 banktrade.xml，banktrade.xml 的内容如下所示：

```
<Context docBase=" D:\ banktrade_code\WebRoot
        privileged="true"   antiResourceLocking="false"   antiJARLocking=
"false">
</Context>
```

其中，D:\ banktrade_code\是前面步骤创建的目录。Tomcat 启动时会读取该文件。

5．启动服务

本例中，Tomcat 安装路径是 C:\Program Files\Apache Software Foundation\tomcat6.0。

■“网银子系统”的启动

在操作系统命令行中执行命令：

```
C:\Program Files\Apache Software Foundation\tomcat6.0/bin/startup.bat
```

■“网银子系统”的关闭

在操作系统命令行中执行命令：

```
C:\Program Files\Apache Software Foundation\tomcat6.0/bin/shutdown.bat
```

6．验证系统是否成功运行

打开 IE 浏览器，输入地址 http://localhost:8080/banktrade 并按回车键，如果出现如图 24-1 所示的界面，说明系统发布成功。

24.7 代码解释

本节将对“网银子系统”的部分重要代码进行解释。

24.7.1 安装 Eclipse

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台，主要用于 Java 开发。Eclipse 工具使用起来简单、方便，比较适合刚入门的朋友使用。

从 http://www.eclipse.org 下载 Eclipse 的压缩包，解压后直接运行 Eclipse.exe 就能进入 Eclipse 的主界面。

24.7.2 Eclipse 导入项目

为了使读者能更清楚地查看“网银子系统”的代码以及代码之间的层次结构关系，我们将项目导入到 Eclipse 中，具体操作步骤说明如下。

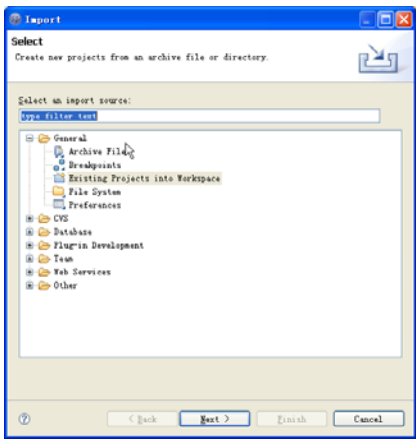


图 24-33 Import 界面

STEP 01 打开 Eclipse，单击“File”菜单，选择“Import”选项，出现如图 24-33 所示的界面。

STEP 02 选择“Existing Projects into Workspace”选项，单击“Next”按钮，出现如图 24-34 所示的界面。

STEP 03 选中“Select root directory”，单击“Browse”按钮，找到“网银子系统”项目所在的目录（本例中是 D:\ banktrade_code\），最后单击“Finish”按钮，项目导入完成，如图 24-35 所示：

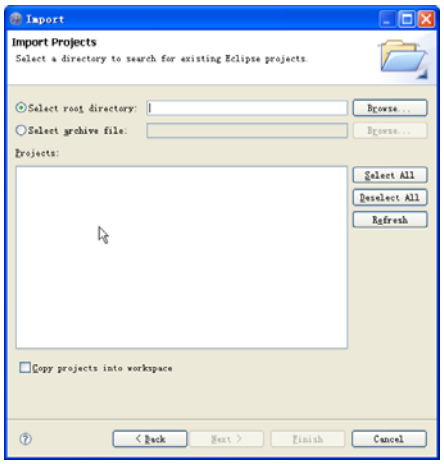


图 24-34　选择目录

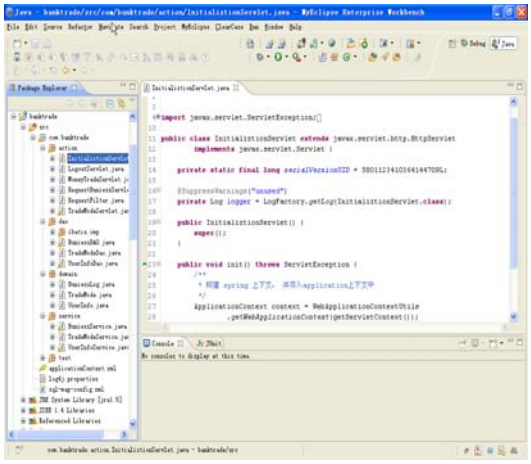


图 24-35　层次展示

24.7.3　MVC 框架

“网银子系统”基于 MVC 框架，MVC 英文即 Model-View-Controller，即把一个应用的输入、处理、输出流程按照 Model、View、Controller 的方式进行分离（如图 24-36 所示），这样一个应用被分成三个层——模型层（Model）、视图层（View）、控制层（Controller）。

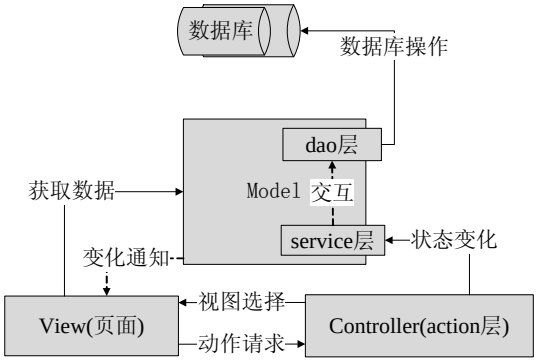


图 24-36　MVC 框架

1．Model 层

就是业务流程（状态）的处理以及业务规则的制定。com/banktrade/service 层和 com/banktrade/dao 层同属于模型层范围，com/banktrade/service 层负责和 com/banktrade/dao 层中具体的 dao 类通信，并对 com/banktrade/action 层提供了具体的业务操作对象，com/banktrade/dao 层封装了与后台数据库交互的各个 dao 类。

2．View 层

View 层（视图层）代表用户交互界面。视图层包含了 WebRoot 目录下所有的 JSP 页

面，其主要用于前台的效果展示以及用户进行各种操作的入口。

3．Controller 层

Controller 层（控制层）可以理解为从用户处接收请求，将模型与视图匹配在一起，共同完成用户的请求。com/banktrade/action 层中所有的 servlet 类承担了控制转发的角色，com/banktrade/action 层主要负责接收用户请求、获取请求中部分信息、控制请求的转发以及最终应该跳转到哪个页面显示效果。

24.7.4 重要代码讲解

本节将节选重要代码进行解释。

1．用户登录

用户通过客户端向系统服务端发起登录请求，服务端获取客户端发来的请求和请求参数，通过 service 层到后台验证用户信息是否合法，如果不合法则将跳转到登录界面并提示登录失败的原因，否则跳转到系统主页，如图 24-37 所示。

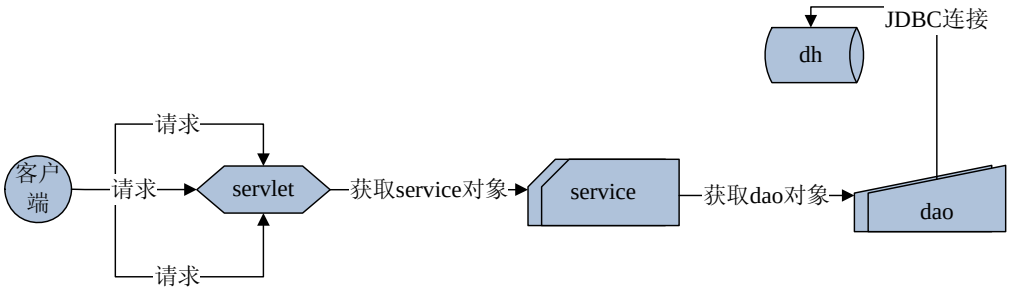


图 24-37　用户登录流程

用户登录功能重要代码位于 com\banktrade\action 层 RequestBuniessServlet 类，具体如下所示：

```
    public  void  service(HttpServletRequest  request,  HttpServletResponse
response)throws ServletException, IOException {
            UserInfo
uinfo=(UserInfo)request.getSession().getAttribute("user");
    //从 session 中获取用户信息，判断用户是否已经登录
            if(uinfo == null){ //用户第一次登录时，获取用户在登录界面上填写的一些信息
    String cardId = request.getParameter("cardID");
    String pwd = request.getParameter("password");
    String num = request.getParameter("PT_CONFIRM_PWD");

    /**
    * 判断用户输入的验证码是否正确
```

```
    */
    String random = (String)request.getSession().getAttribute("random");
        if(num == null || random == null || !num.equals(random)){
            request.setAttribute("messenger", "验证码输入错误");
                    RequestDispatcher
dis=request.getRequestDispatcher("login.jsp"); //验证失败，则跳转到登录界面
                    dis.forward(request,response);
                    return ;
            }
                context               =               (ApplicationContext)
getServletContext().getAttribute("context");
                UserInfoService                  service                  =
(UserInfoService)context.getBean("userInfoService");
                UserInfo user = service.getUserInfo(cardId,"",pwd);
                if(user == null){
                        request.setAttribute("messenger", "登录失败 请检查用户名
和密码是否正确");
                        RequestDispatcher
dis=request.getRequestDispatcher("login.jsp"); //用户不存在，则跳转到登录界面
                        dis.forward(request,response);
                        return ;
                }
                //把用户放进 Session 中
                request.getSession().setAttribute("user", user);
                request.getSession().setAttribute("num", num);
                //跳转到用户账号信息页面,展现用户账号信息
                request.setAttribute("user", user);
        }
            RequestDispatcher dis= request.getRequestDispatcher("index.jsp");
//登录成功
            dis.forward(request,response);
            return;
        }
```

2．账户信息展示

账户信息展示页面主要显示账户的一些基本信息，并提供 RMB 交易和交易明细查询等功能，如图 24-38 所示。

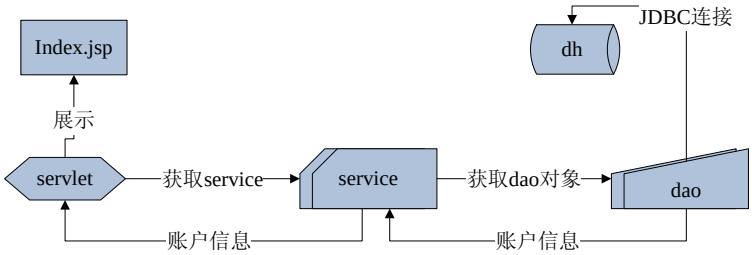


图 24-38　账户信息展示流程

具体请参考本书配套光盘下的 banktrade\banktrade_code\WebRoot 目录下的 index.jsp 文件。

3．RMB 交易

RMB 交易功能可以实现用户存、取款操作。交易成功后，系统更改用户余额，同时还会刷新 index.jsp 页面上的用户余额，如图 24-39 所示。

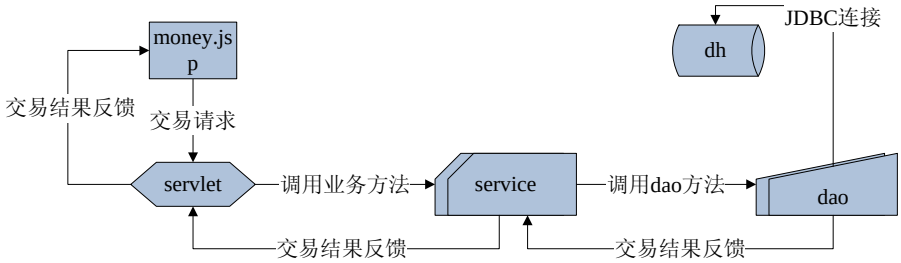


图 24-39　RMB 交易流程

RMB 交易功能的重要代码位于 com\banktrade\action 层 MoneyTradeServlet 类，具体介绍如下：

```
    public  void  service(HttpServletRequest  request,  HttpServletResponse
response)
        throws ServletException, IOException{
            context = (ApplicationContext) getServletContext().getAttribute
("context");
            BuniessService   bservice   =   (BuniessService)context.getBean
("buniessService");
            UserInfoService   uservice   =   (UserInfoService)context.getBean
("userInfoService");
            /**
            * 获取用户交易信息
            */
            String money = request.getParameter("money").trim();
            String type = request.getParameter("type");
            String mode = request.getParameter("mode");
```

```
            UserInfo   user   =   (UserInfo)request.getSession().getAttribute
("user");
            //取钱之前先判断该用户账户余额是否大于要去的钱数
            if(type.equals("1")){
               UserInfo userNew = uservice.getUserInfo(user.getCardId(),user.
getPId(),user.getPassword());
                double ymoney = userNew.getMoney();
                double my = Double.parseDouble(money);
                if(ymoney < my){
                    response.sendRedirect("fail.jsp");
                    //交易失败,跳转到fail.jsp页面
                    request.setAttribute("messenger","交易失败");
                    return;
                }
                //取钱
                balance = ymoney-my;
                int mflag = uservice.getMoney(user.getCardId(), balance);
                if(mflag == 0){
                    response.sendRedirect("fail.jsp");
                    //交易失败,跳转到fail.jsp页面
                    request.setAttribute("messenger","交易失败");

                    return;
                }
            }else{   //存钱
                UserInfo  userNew  =  uservice.getUserInfo(user.getCardId(),
user.getPId(),user.getPassword());
                double ymoney = userNew.getMoney();
                double my = Double.parseDouble(money);
                my = my + ymoney;
                int flag = uservice.saveMoney(user.getCardId(), my);
                if(flag == 0){
                    response.sendRedirect("fail.jsp");
                    //交易失败,跳转到fail.jsp页面
                    request.setAttribute("messenger","交易失败");

                    return;
                }
            }
            //money.jsp页面上应判断用户输入的钱数是否合法

            BuniessLog log = new BuniessLog();
            log.setModeId(Integer.parseInt(mode));
            log.setMoney(Double.parseDouble(money));
            if(Integer.parseInt(type) == 1){
                //支出
```

```
                log.setType(0);
            }else{
                log.setType(1);
            }

            log.setTradeTime(new Date());

            log.setCardId(user.getCardId());
            log.setId(图 25 );
            log.setBalance(balance);

            //添加交易日志和该用户账户余额更新是一个原子操作
            int flag = bservice.addBuniessLog(log);
            if(flag == 0){
                response.sendRedirect("fail.jsp");
                request.setAttribute("MoneyTradeMessenger","0");
                return;
            }
            //设置最新的 session 中的用户
            UserInfo    user1    =    uservice.getUserInfo(user.getCardId(),
user.getPId(), user.getPassword());
            request.getSession().setAttribute("user", user1);
            request.setAttribute("MoneyTradeMessenger","1");
            RequestDispatcher dis=request.getRequestDispatcher("money.jsp");

            dis.forward(request,response);
            return;
        }
```

4．交易明细查询

交易明细的功能是：只需选择时间段，用户便可查询在某个时间段内的交易明细，如图 24-40 所示。

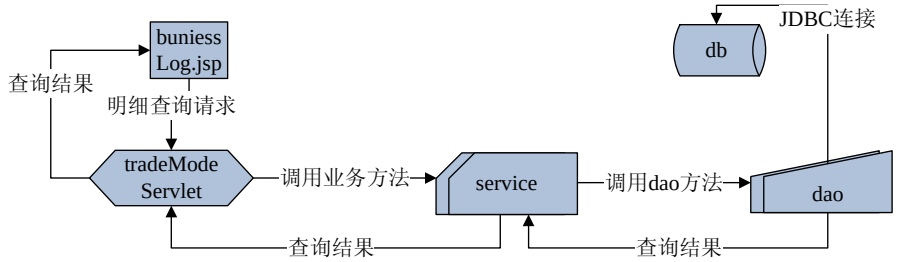


图 24-40　交易明细查询流程

交易明细功能重要代码位于 com\banktrade\action 层中 TradeModeServlet 类，具体介绍如下：

```
    public  void  service(HttpServletRequest  request,  HttpServletResponse
response)
        throws ServletException, IOException{
            //判断用户是否登录
            UserInfo  user  =  (UserInfo)request.getSession().getAttribute
("user");
            //获取用户选择的时间范围
            String time = request.getParameter("demo");
            String endTime = request.getParameter("demo1");
            //判断时间是否合法
            if(time == null || time.equals("")){
                time = "0000-00-00 00:00:00";
            }
            if(endTime == null || endTime.equals("")){
                endTime = "0000-00-00 00:00:00";
            }
            java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat(
            "yyyy-MM-dd hh:mm:ss");

            Date beginTradeTime = null;
            Date endTradeTime = null;
            try {  //构造时间范围
                beginTradeTime = sdf.parse(time+" 00:00:00");
                endTradeTime = sdf.parse(endTime+" 23:59:59");
            } catch (ParseException e) {
                e.printStackTrace();
            }

            context = (ApplicationContext) getServletContext().getAttribute
("context");
            TradeModeService  service  =  (TradeModeService)context.getBean
("tradeModeService");
            BuniessService   bservice   =   (BuniessService)context.getBean
("buniessService");
            //获取用户所选的范围内的交易明细
           List log = bservice.getBuniessLog(user.getCardId(),beginTradeTime,
endTradeTime, -1, null);
            Map tradeLog = null;
            if( log != null){
                tradeLog = new HashMap();
                for(int i=0;i<log.size();i++){
                    BuniessLog blog = (BuniessLog)log.get(i);
                    TradeMode tradeAddress = service.getTradeModeInfo(blog.
getModeId());
                    //Map 封装交易 ID 和 交易地点
                    tradeLog.put(tradeAddress.getId(),tradeAddress.
```

```
getBuniessAddress());
                }
            }

            request.setAttribute("buniessLogs",log);
            //交易 ID 和交易地点 最好能封装到 Map 中，以便页面上能直接获取 而不用生成新对象

            request.setAttribute("tradeModeLogs", tradeLog);

            if( user == null){
                RequestDispatcher dis= request.getRequestDispatcher("fail. jsp");
                dis.forward(request,response);
                return;
            }

            RequestDispatcher dis= request.getRequestDispatcher("businessLog. jsp");
            dis.forward(request,response);
            return;
        }
```

以上只是对部分代码进行了简单的介绍，具体代码请参考本书配套光盘中的 banktrade 目录。



本章总结

本章按照软件生命周期（SDLC）对“网银子系统”进行了介绍，并给出详细的实施步骤，使读者能够独立在自己的机器上搭建一个“网银子系统”。本章的目的是让读者理解整个软件开发的生命周期，并熟悉 Java 开发的大致流程。由于篇幅所限，我们只能把读者领到这里，希望各位朋友在以后的学习中不畏艰辛，勇于探索，总有一天，你一定能够成为一个优秀的开发人员！

《成功之路：Oracle 11g 学习笔记》读者交流区

尊敬的读者：

感谢您选择我们出版的图书，您的支持与信任是我们持续上升的动力。为了使您能通过本书更透彻地了解相关领域，更深入的学习相关技术，我们将特别为您提供一系列后续的服务，包括：

1．提供本书的修订和升级内容、相关配套资料；

2．本书作者的见面会信息或网络视频的沟通活动；

3．相关领域的培训优惠等。

请您抽出宝贵的时间将您的个人信息和需求反馈给我们，以便我们及时与您取得联系。

您可以任意选择以下三种方式与我们联系，我们都将记录和保存您的信息，并给您提供不定期的信息反馈。

1．短信

您只需编写如下短信：B10781+您的需求+您的建议

发送到1066 6666 789（本服务免费，短信资费按照相应电信运营商正常标准收取，无其他信息收费）

为保证我们对您的服务质量，如果您在发送短信24小时后，尚未收到我们的回复信息，请直接拨打电话（010）88254369。

2．电子邮件

您可以发邮件至jsj@phei.com.cn或editor@broadview.com.cn。

3．信件

您可以写信至如下地址：北京万寿路173信箱博文视点，邮编：100036。

如果您选择第2种或第3种方式，您还可以告诉我们更多有关您个人的情况，及您对本书的意见、评论等，内容可以包括：

（1）您的姓名、职业、您关注的领域、您的电话、E-mail地址或通信地址；

（2）您了解新书信息的途径、影响您购买图书的因素；

（3）您对本书的意见、您读过的同领域的图书、您还希望增加的图书、您希望参加的培训等。

如果您在后期想退出读者俱乐部，停止接收后续资讯，只需发送“ B10781+退订”至10666666789即可，或者编写邮件“ B10781+退订+手机号码+需退订的邮箱地址”发送至邮箱：market@broadview.com.cn 亦可取消该项服务。

同时，我们非常欢迎您为本书撰写书评，将您的切身感受变成文字与广大书友共享。我们将挑选特别优秀的作品转载在我们的网站（www.broadview.com.cn）上，或推荐至CSDN.NET等专业网站上发表，被发表的书评的作者将获得价值50元的博文视点图书奖励。

我们期待您的消息！
博文视点愿与所有爱书的人一起，共同学习，共同进步！

通信地址：北京万寿路 173 信箱　博文视点（100036）　　电话：010-51260888
E-mail：jsj@phei.com.cn，editor@broadview.com.cn

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888
传　　真：（010）88254397
E-mail：　dbqq@phei.com.cn
通信地址：北京市万寿路 173 信箱
　　电子工业出版社总编办公室
邮　　编：100036